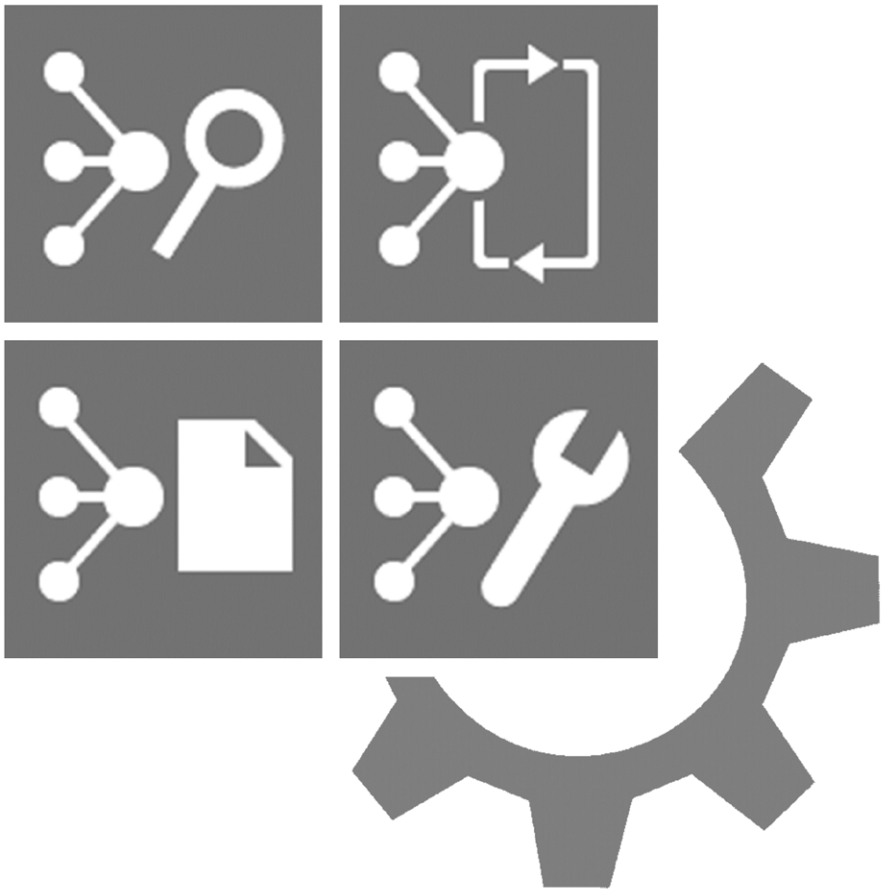
RENISHAW
apply innovation™

Manual del desarrollador de DataHUB



Esta página se ha dejado intencionadamente en blanco.

RENISHAW
apply innovation™

Índice

1 Antes de empezar

1.1 Términos y condiciones y garantía

A no ser que usted y Renishaw hayan acordado y firmado un contrato independiente por escrito, el equipo y el software se venden a tenor de los Términos y Condiciones Generales de Renishaw, que se facilitan con dicho equipo o software o están disponibles previa petición en su oficina local de Renishaw.

Renishaw garantiza sus equipos y software durante un período limitado (según se establece en nuestros Términos y condiciones estándar) si se ha instalado exactamente tal como se define en la documentación de Renishaw relacionada. Consulte los Términos y condiciones estándar para conocer los detalles de la garantía.

El equipo y el software adquirido a terceros proveedores se regirá por términos y condiciones independientes facilitados junto a dicho equipo y software. Para obtener más información, consulte a su proveedor.

1.2 Cambios del equipo

Renishaw se reserva el derecho de realizar modificaciones a las especificaciones sin previo aviso.

1.3 Patentes

Las características de la máquina de fabricación aditiva y de otros sistemas similares de Renishaw están sujetas a una o varias de las siguientes patentes y aplicaciones de patentes

1.3.1 Serie RenAM 500 (modelos Q, S y Flex)

CA 2738618	EP 2331232	IN WO2014/125258	US 10335901
CA 2738619	EP 2875855	IN WO2014/125280	US 10493562
	EP 2956261	IN WO2014/199134	US 10500641
CN 102186554	EP 2956262		US 10639879
CN 105102160	EP 3007879	JP 6482476	US 10933620
CN 105228775	EP 3221073	JP 6571638	US 10974184
CN 105492188	EP 3221075		US 11033968
CN 107107193	EP 3299110		US 11040414
CN 107206494	EP 3323534		US 11104121
CN 107921659	EP 3325240		US 11267052
CN 108189390	EP 3357606		US 11305354
CN 108349005	EP 3377252		US 11478856
CN 108515182	EP 3377253		US 11565346
CN 109177153	EP 3566798		US 8753105
	EP 3689507		US 8794263
	EP 4023387		US 9114478
			US 9669583
			US 9849543
			US 2020-0023463
			US 2021-0354197
			US 2022-0203451
			US 2023-0122273

1.3.2 DataHUB

CN 109937101	EP 3482855	US 11167497	WO 2020/099852
CN 111315512	EP 3538295	US 2020-0276669	
CN 112996615	EP 3880391	US 2021-0394272	

1.3.3 InfiniAM Spectral

CN 105745060	EP 3049235	US 10850326	WO 2020/099852
CN 108349005	EP 3377252	US 11305354	WO 2020/174240
CN 109937101	EP 3482855	US 11040414	
CN 110026554	EP 3482909	US 2020-0276669	
CN 111315512	EP 3538295	US 2021-0039167	
CN 111491777	EP 3880391	US 2021-0394272	
CN 112996615	EP 3930999	US 2022-0168813	
CN 115943048	EP 2020-174240	US 2022-0203451	

2 Introducción

2.1 Alcance del suministro

Este documento es una guía para la construcción, ensayo y desarrollo de un componente de software de plug-in para DataHUB. Para ver la descripción general y una explicación de las funciones del software DataHUB, consulte la guía de usuario de DataHUB (n.º de referencia Renishaw H-5800-6854). El ciclo de desarrollo del plug-in se inicia al instalar DataHUB en el PC de desarrollo y, a continuación, se crea en Visual Studio un conjunto .NET que contiene el código del plug-in. Si Visual Studio está configurado correctamente para la depuración de errores, puede probar y corregir errores del plug-in para verificar si los valores son correctos antes de la instalación final en un PC de recopilación de datos (DCPC) en una máquina de FA de producción.

2.2 Siglas

Término	Definición
AM	Fabricación aditiva
AMPM	Control de procesos de fabricación aditiva
DCPC	Ordenador personal de recopilación de datos
HMI	Interfaz hombre máquina (pantalla táctil)
OEM	Fabricante de equipos originales
PC	Ordenador personal
PLC	Controlador lógico programable
REACH	Registro, Evaluación, Autorización y Restricción de sustancias químicas
WEEE	Eliminación de residuos de equipos eléctricos y electrónicos

2.3 Información de seguridad de esta guía de usuario

En esta guía, la información adicional importante se resalta con una Advertencia, Precaución o Nota. A continuación se muestran ejemplos de las definiciones.

2.3.1 Advertencia

Un ejemplo de una advertencia es el siguiente:

ADVERTENCIA: Una advertencia indica al usuario que, si no se siguen las instrucciones indicadas, existe riesgo de lesiones para el operario y otras personas próximas.

2.3.2 Precaución

Un ejemplo de una precaución es el siguiente:

PRECAUCIÓN: Precaución indica al usuario que, si no se siguen las instrucciones indicadas, existe riesgo de dañar el equipo.

2.3.3 Nota

Un ejemplo de una nota es el siguiente:

NOTA: Una Nota advierte al usuario de la importancia de la información relacionada, o le asesora sobre la tarea o actividad que está realizando.

2.4 Programa de formación

Renishaw proporciona un nivel básico de formación para operar DataHUB de forma segura. Renishaw también ofrece cursos de formación ampliados para operarios e ingenieros de proceso. Consulte la guía de usuario de DataHub (n.º de referencia Renishaw H-5800-6854) y la guía de formación incluidas en el curso de formación del usuario que todos los usuarios deben completar antes de utilizar el DataHUB.

2.5 Documentación de referencia

Además de esta guía de usuario, consulte también los siguientes documentos de información adicional sobre otros aspectos del paquete integrado InfiniAM® y las máquinas de FA de Renishaw.

- Guía de instalación de la máquina de fabricación aditiva RenAM 500Q/S (n.º de referencia Renishaw H-5800-3692)
- Guía de usuario de la máquina de fabricación aditiva de RenAM 500Q/S (n.º de referencia Renishaw H-5800-3693)
- Guía de instalación del software InfiniAM® y DataHUB (n.º de referencia Renishaw H-5800-6846)
- Guía de usuario de DataHUB (n.º de referencia Renishaw H-5800-6854)
- Guía de usuario de InfiniAM® Spectral (n.º de referencia Renishaw H-5800-6842)
- Guía de usuario de InfiniAM® Camera (n.º de referencia Renishaw H-5800-6850)
- Guía de usuario del Administrador de licencias de Renishaw v2.x (que puede descargar mediante el enlace incluido en el correo electrónico de la licencia de InfiniAM)
- Plug-in Export Packets (n.º de referencia Renishaw 5800-6788)

3 Piezas de repuesto

DataHUB no tiene piezas que precisen mantenimiento del usuario. Si se produce algún fallo en DataHUB, la reparación consiste en la reinstalación y configuración del software.

Para programar una visita del servicio técnico, consulte los datos de contacto de su oficina local de Renishaw en la Sección 4, “Datos de contacto”.

El software InfiniAM y DataHUB se actualiza periódicamente. Los usuarios registrados pueden descargar la última versión del software en su cuenta MyRenishaw.com.

Esta página se ha dejado intencionadamente en blanco.

4 Datos de contacto

Número de teléfono	+34 93 6633420	
Horario de atención	De lunes a jueves	De 8:00 a 17:00 (CEST)
	Viernes	De 8:00 a 16:00 (CEST)
Correo electrónico	SM_ES_BAR_AM_SUPPORT_IBE@Renishaw.com	
Dirección del servicio técnico	Renishaw Ibérica S.A.U. Gavà Park C. de la Recerca, 7 08850 GAVÀ Barcelona España	
Tipo de sistema de FA		
Número de serie de la máquina de FA		
Números de versión del software	Versión de HMI	
	Versión de PLC	
	Versión de PC	
Número de serie del hardware InfiniAM Spectral (módulo MeltVIEW)		
Número de versión del software InfiniAM		
Número de versión del software DataHUB		

Indique los detalles expuestos anteriormente. Puede consultar los detalles de la máquina de FA de Renishaw en la placa de características de la parte posterior de la máquina. Puede consultar los detalles del módulo MeltVIEW en la placa de características visible cuando el módulo InfiniAM está instalado en la máquina de FA de Renishaw. Puede consultar los detalles de CameraVIEW en la etiqueta de la parte posterior de la máquina. El hardware CameraVIEW se encuentra detrás de una tapa encima de la cámara.

Si precisa asistencia adicional, póngase en contacto con su oficina local de Renishaw. Consulte:

www.renishaw.es/contacto

Esta página se ha dejado intencionadamente en blanco.

5 Seguridad

5.1 Introducción

ADVERTENCIA: La información de seguridad corresponde a las guías de usuario e instalación de la máquina de FA de Renishaw, salvo que se indique lo contrario en este documente.

ADVERTENCIA: Antes de conectar el láser de la máquina de FA, compruebe que el módulo de hardware MeltVIEW está instalado en la abertura láser.

ADVERTENCIA: El módulo de hardware MeltVIEW no tiene sistema de bloqueo conectado con el circuito de seguridad láser. Si el módulo de hardware MeltVIEW está retirado y se conecta el láser, se emite una luz láser perjudicial a través de la abertura del láser de la máquina de FA

5.2 Etiquetas de advertencia y láser específicas de DataHUB Generator

La máquina de FA no lleva etiquetas de advertencia o seguridad específicas del sistema DataHUB Generator.

Esta página se ha dejado intencionadamente en blanco.

6 Operación

6.1 Introducción

Mediante esta arquitectura de plug-in, el paquete integrado de software InfiniAM proporciona análisis en tiempo real de los datos de control de procesos obtenidos en las plataformas de hardware LaserVIEW y MeltVIEW. Un plug-in es una unidad de software independiente instalada en un PC de recopilación de datos (DCPC) en el que se ejecuta DataHUB. Cuando DataHUB procesa los datos obtenidos de una construcción de fabricación aditiva, transfiere los valores de control del proceso a los plug-in para que estos realicen los análisis personalizados. DataHUB permite ejecutar simultáneamente varios plug-in, por puede aplicar diversos algoritmos y técnicas de análisis a los datos de control de procesos, además, está integrado en el sistema Renishaw Central, de forma que los plug-in pueden mostrar resultados de análisis junto con otras lecturas de los sensores de máquina obtenidos durante la construcción.

DataHUB incluye dos versiones de API de plug-in: los plug-in de la versión 1.0 (V1.0) procesan los datos de construcción agrupados en paquetes (consulte “Muestras a paquetes” en la página 6-8); los plug-in de la versión 2 (V2.0) procesan los datos de construcción como secuencias de código (consulte “Muestras a secuencia de código” en la página 6-8). Es importante entender que API V2.0 no sustituye a API V1.0, puesto que muestran los mismos datos de control de procesos, aunque en distintos formatos. Debe estudiarse cuidadosamente la elección de API conforme a los análisis que se van a realizar: los paquetes de la versión V1.0 proporcionan un número de muestras significativamente menor que la versión V2.0 y, puesto que no todas las tareas de análisis necesitan datos de muestra de alta precisión, pueden ser más adecuados.

En este documento se detalla la arquitectura del sistema y el ciclo de vida del plug-in, así como otros aspectos del sistema correspondientes a las dos versiones de API.

El plug-in "Export Packets" crea un archivo delimitado por tabuladores que contiene un resumen de los datos del proceso obtenidos por el hardware LaserVIEW y MeltVIEW durante una construcción. Este plug-in se entrega con DataHUB y no requiere licencia adicional.

6.2 Arquitectura

Las relaciones de nivel superior entre los elementos de hardware y software del sistema se muestran en la Imagen 1:

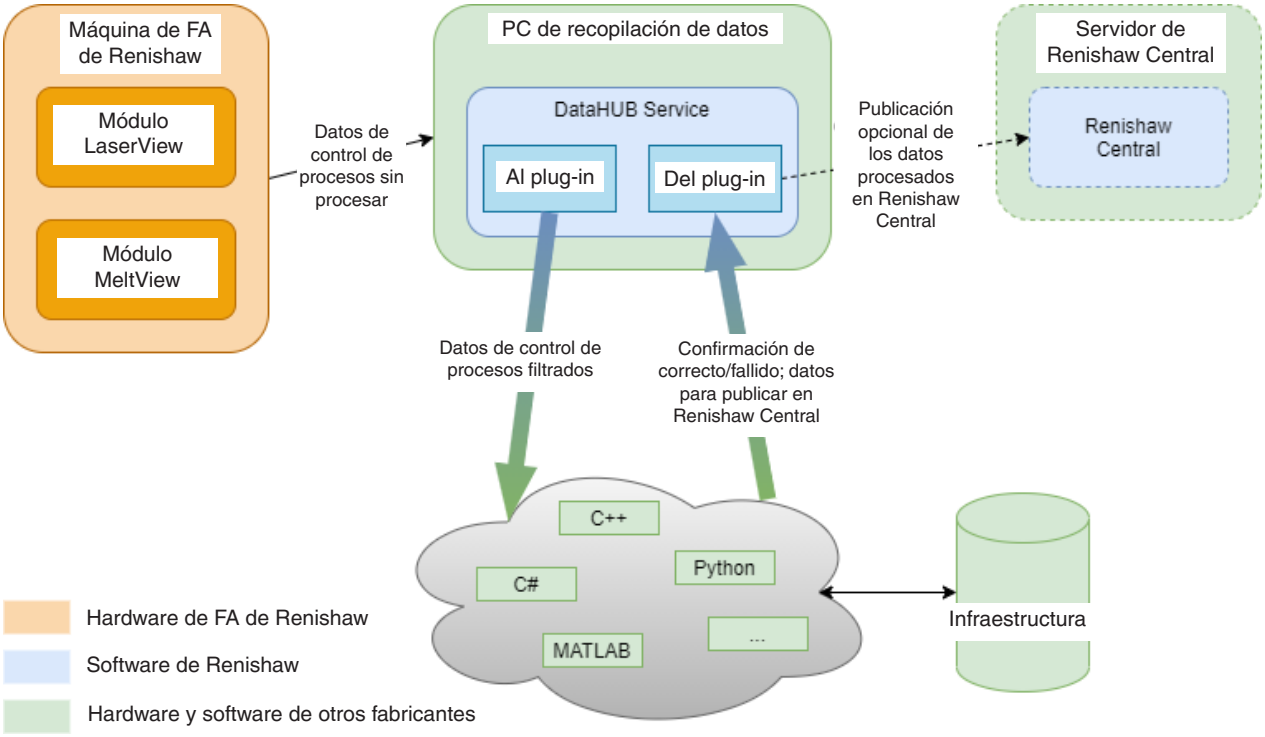


Imagen 1 Arquitectura del plug-in DataHUB

6.2.1 Caudal de datos

Durante la ejecución de una construcción en una máquina de FA de Renishaw equipada con el hardware LaserVIEW y MeltVIEW, se transfieren los datos de control del proceso al DCPC. El servicio DataHUB en ejecución en el DCPC distribuye los datos a todos los plug-in instalados. DataHUB ejecuta una instancia de cada plug-in instalado por cada construcción. Cada instancia tiene asignada una carpeta de salida con nombre exclusivo para cada resultado persistente, ejecutadas independientemente, para garantizar que el fallo de una instancia no provoca un fallo general del sistema.

Imagen 2 muestra el caudal de datos típico durante el proceso de construcción:

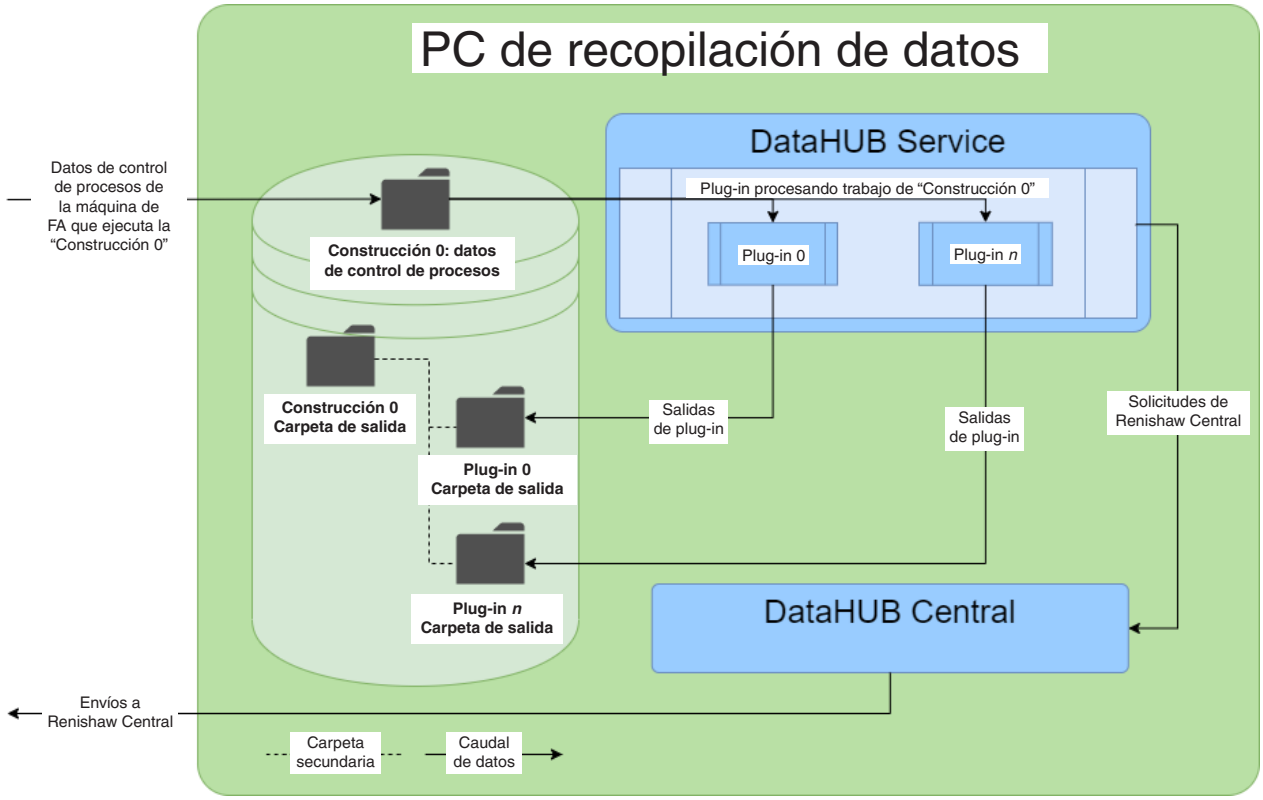


Imagen 2 Caudal de datos de construcción

Los datos de control del proceso obtenidos en una máquina de FA durante la construcción se envían a una carpeta del DCPC. El servicio DataHUB supervisa esta carpeta y, cuando recibe nuevos datos, los deriva a las instancias del plug-in asociado al trabajo de construcción. Los plug-in pueden notificar los resultados al servicio DataHUB (datos de tiempo, alertas y resultados de análisis) que se deben transmitir a Renishaw Central. El servicio DataHUB Central gestiona estas peticiones de envío.

6.2.2 Tecnología de análisis de plug-in

DataHUB no confía mucho en la tecnología empleada para instalar los plug-in y la infraestructura necesaria. Sus dos requisitos son los siguientes:

1. Un plug-in es un conjunto .NET que instala una serie de métodos definidos como API de plug-in.

2. Las dependencias de tiempo de ejecución se instalan en el DCPC.

DataHUB invoca los métodos API de plug-in con los valores completados con los datos de control del proceso y, a continuación, la instalación del plug-in tiene la libertad para pasar los datos a la tecnología que mejor se adapte a la tarea de análisis más próxima. En este documento no se trata en detalle la integración .NET con otras tecnologías y lenguajes de programación, pero puede obtener gran cantidad de información de dominio público.

6.2.3 Caudal de datos “Push”

El servicio DataHUB envía “push” los datos a los plug-in: las API de plug-in API no interrogan los juegos de datos ni extraen valores de datos. Su uso típico es la ejecución de análisis en tiempo real a medida que progresa la construcción. Para juegos de datos archivados, puede utilizar DataHUB Generator para instruir al servicio DataHUB que reprocese los datos de forma que se pueda realizar el análisis adicional de plug-in.

6.2.4 Vida útil del plug-in

Para utilizar un plug-in en DataHUB, deben validarse previamente los requisitos de instalación de la API, por tanto, solo si son conformes, se crean y utilizan las instancias para analizar los datos de la construcción. Una vez validada, se crea una instancia del plug-in por cada nueva tarea de procesamiento de datos de construcción. Las fases de la vida útil de un plug-in se muestran en la Imagen 3:

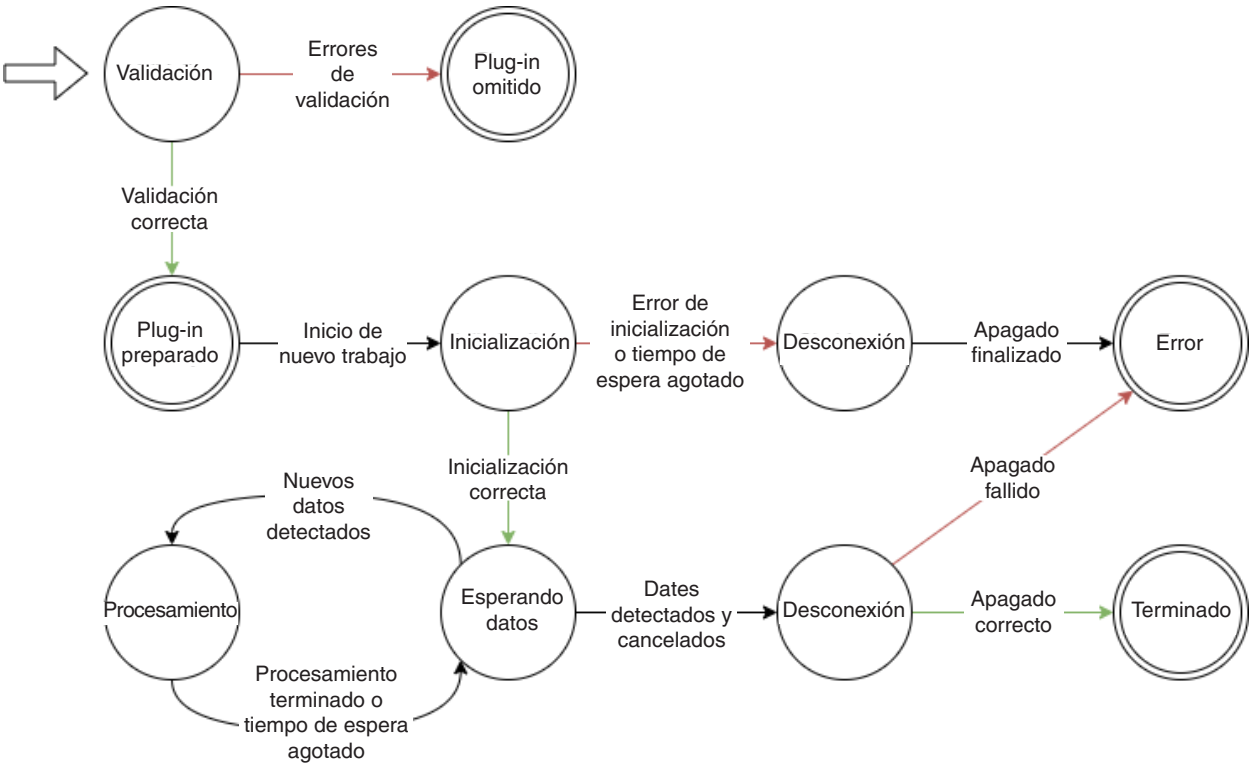


Imagen 3 Los distintos estados de un plug-in pueden servir durante toda su vida útil

6.2.4.1 Validación

Cuando se inicia DataHUB, busca los conjuntos .NET que instalan una o varias clases públicas conformes con las API de plug-in DataHUB. Todos los que son conformes se registran como plug-in válidos y se crean las instancias cada vez que se inicia un trabajo en “Procesar LaserVIEW y MeltVIEW mediante plug-in”.

Un conjunto puede contener instalaciones de más de un plug-in, si es necesario, incluidas distintas versiones de API.

6.2.4.2 Inicialización

Cuando DataHUB inicia un trabajo de plug-in, se crea una nueva instancia de un plug-in y se *Inicializa* el método invocado. Los valores que se pasan a este método son “constantes de construcción”, por ejemplo, el número de láseres sobre los que el plug-in puede calcular los requisitos de asignación de recursos, etc. Este método devuelve información sobre si se ha inicializado correctamente o no y, en caso de fallo, se invoca inmediatamente la instancia del método de *Shutdown*. Opcionalmente, el plug-in podría devolver definiciones de series cronológicas de ejes que se rellenarán con puntos de datos generados durante el procesamiento posterior (consulte “Publicación de datos en Renishaw Central” en la página 6-6).

6.2.4.3 En espera de datos/procesando

El método de procesamiento de datos se invoca las veces que sea necesario para enviar todos los datos de control del proceso de una construcción a una instancia de plug-in. Por cada llamada, el plug-in devuelve información sobre si ha procesado o no los datos correctamente. Si indica un error, el plug-in no entra en modo de estado de error, sino que continúa recibiendo datos (el supuesto de si los datos son de otras capas depende del plug-in). Opcionalmente, el plug-in podría devolver series cronológicas de puntos de datos, alertas y reenviar otros análisis detectados a Renishaw Central (consulte “Publicación de datos en Renishaw Central”).

6.2.4.4 Desconexión

El método de *Shutdown* se invoca cuando no quedan datos para procesar (es decir, la construcción se ha completado o se ha cancelado). En este método, el plug-in debe realizar el procesamiento final de los datos de construcción, dejar libres los recursos utilizados, etc.

6.2.5 Duración

La duración se comunica en la API en microsegundos respecto al inicio de una capa. Una duración de 0 coincide con el inicio de una capa, si es de 1000, representa 1000 microsegundos después de iniciar la capa, etc. Si es necesario (por ejemplo, para la transferencia a Renishaw Central) DataHUB considera los tiempos relativos devueltos por un plug-in en un plazo de tiempo absoluto, como UTC.

6.2.6 Publicación de datos en Renishaw Central

Si DataHUB está conectado a Renishaw Central, un plug-in puede publicar resultados de análisis a través de DataHUB. Si la cadena devuelta por los métodos *Initialise* y Process es conforme con el esquema JSON registrado en las definiciones de la API (consulte la Sección 6.4, “Plug-in API V2”), DataHUB transfiere los datos que contiene al punto final de Renishaw Central.

DataHUB reconoce tres tipos de transferencias a Renishaw Central:

- Alertas
- Resultados del análisis
- Serie cronológica

6.2.6.1 Alertas

Una alerta indica que ha ocurrido un evento importante en un momento determinado. Un plug-in puede generar una alerta si detecta una anomalía en los datos de construcción, por ejemplo, si los datos indican que ha ocurrido un fallo o va a ocurrir pronto en la construcción. Hay tres niveles de alertas: *Información*, *Advertencia* y *Error*, y se utilizan para informar a la aplicación de visualización (como el sitio web de Renishaw Central) sobre el nivel de importancia para generar una alerta.

Las alertas solo se generan durante la fase de duración del *Proceso*.

6.2.6.2 Resultados del análisis

Los resultados de análisis, como alertas, se pueden devolver con relación al análisis a realizado en los datos de producción. La definición es flexible y permite especificar el contenido según el tipo de resultado. Esto significa que, por ejemplo, si el sitio web de Renishaw Central no sabe cómo representar todas las facetas de un análisis de resultados, puede escribir una aplicación a medida para hacerlo.

Los resultados del análisis solo se generan durante la fase de duración del *Proceso*.

6.2.6.3 Serie cronológica

Una serie cronológica representa una serie de valores durante un tiempo. Puede contener cualquier tipo de datos, siempre que puedan imprimirse con el tiempo. Durante la fase de *Proceso*, un plug-in puede añadir dos tipos de datos a una serie cronológica: valores y límites. Los valores se definen en parejas de tiempo/valor que añade un punto de datos a la serie cronológica. Los límites definen campos esperados para facilitar la visualización e indican una situación crítica de los datos.

Las series cronológicas se diferencian de las alertas y los resultados de análisis en que deben definirse antes de añadir puntos de datos. Un plug-in puede devolver definiciones de series cronológicas desde el método *Initialise*. Si el plug-in intenta añadir datos más adelante a una serie cronológica que no estaba en la lista de definición inicial (o si las propias definiciones son incorrectas), los datos se rechazan o se pierden.

6.2.7 Registro de auditoría

DataHUB mantiene un juego de registros que almacenan las acciones realizadas, incluidas algunas que podrían utilizarse para diagnosticar problemas del plug-in. Los registros se guardan en la ubicación:

<$PROGRAMDATA$\Renishaw\DataHUB\Audit Logs>

El nombre de los registros se compone del día de su creación con el formato“aaaaMMdd”.

Las entradas principales de interés para un programador de plug-in son las siguientes:

- Durante el inicio, DataHUB registra:
  - números de versión
  - detalles de la licencia
  - validación de los conjuntos de plug-in

- Al iniciar una construcción, DataHUB registra:
  - creación de la instancia de plug-in
  - valores utilizados para inicializar cada instancia de plug-in
  - el resultado del método *Initialise* devuelto

- Durante una construcción, DataHUB registra:
  - resultados no importantes de llamadas a *Processing* del plug-in (para reducir el correo no deseado al inicio de sesión, un plug-in podría devolver un resultado trivial que no se registra)

- Al terminar una construcción, DataHUB registra:
  - el resultado de cada llamada al método *Shutdown*

6.2.8 Pre-procesamiento de datos de control del proceso

Los módulos LaserVIEW y MeltVIEW obtienen datos en una serie de muestras, cada una asociada a un láser específico, una posición de la mesa del polvo y varias lecturas de los sensores en un momento determinado. DataHUB realiza un pre-procesamiento de los datos de control del proceso que están “sin procesar”. En el plug-in V1.0, DataHUB resume las muestras en paquetes, y en el V2.0, los datos de muestra se agrupan en una secuencia de código.

6.2.8.1 Muestras a paquetes

El servicio DataHUB agrupa muestras consecutivas que comparten la misma posición, con el disparo del láser, en un paquete. La hora de inicio de un paquete empieza con la primera muestra entregada, y la hora de la última muestra enviada indica la duración. Los valores de lectura de los sensores LaserVIEW y MeltVIEW y las muestras entregadas son un promedio.

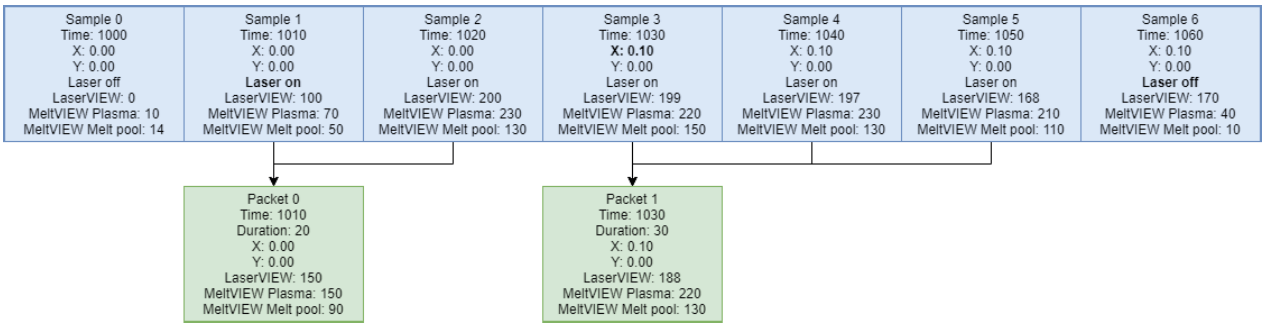


Imagen 4 Muestras en paquetes

En la Imagen 4, se reparten siete muestras en dos paquetes. La primera muestra se omite, ya que el láser no se dispara. Puesto que comparten la misma posición y el láser no se dispara, las muestras 1 y 2 se agrupan en un paquete. La posición de la muestra 3 cambia respecto a la anterior, por tanto, se inicia un nuevo paquete que obtiene muestras hasta que cesan los disparos del láser. La muestra final 6 no se ha incluido porque el láser está apagado, no obstante, comparte la misma posición que la muestra 5.

6.2.8.2 Muestras a secuencia de código

Al convertir las muestras del control de procesos en una secuencia de código, DataHUB genera un código de *Muestras* rodeadas por otro código informativo que, en conjunto, describen la estructura y el contenido de la construcción. Al contrario que en la versión V1.0, se obtienen las muestras cuando el láser está apagado y se añaden a la secuencia de código.

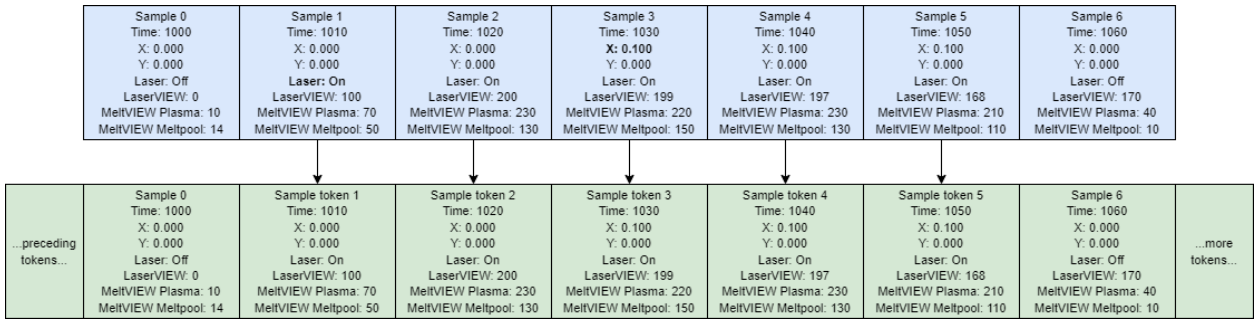


Imagen 5 Muestras a secuencia de código

La Imagen 5 muestra una sección de una secuencia de código siempre que se utilicen las mismas en las *Muestras a paquetes* indicado más arriba.

6.3 Plug-in API V1

Este documento es una guía para la construcción, ensayo y desarrollo de un componente de software de plug-in V1.0 para DataHUB. El ciclo de desarrollo del plug-in se inicia al instalar DataHUB en el PC de desarrollo y, a continuación, se crea en Visual Studio un conjunto .NET que contiene el código del plug-in. Si Visual Studio está configurado correctamente para la depuración de errores, puede probar y corregir errores del plug-in para verificar si los valores son correctos antes de la instalación final en un PC de recopilación de datos (DCPC) en una máquina de FA de producción.

6.3.1 Actividades previas necesarias

Además de este documento, deben leerse los siguientes:

- Guía de instalación del software InfiniAM® y DataHUB (n.º de referencia Renishaw H-5800-4349)
- Guía de usuario de *DataHUB* (n.º de referencia Renishaw H-5800-4761)
- Manual de usuario del Administrador de licencias de Renishaw v2.x

Antes de continuar, debe realizar las siguientes tareas en el PC en el que se van a instalar los plug-in:

- El PC debe disponer de una GPU que cumpla como mínimo la especificación mínima de DataHUB (consulte la especificación de hardware del PC de recopilación de datos en la guía de instalación del software InfiniAM® y DataHUB).
- Compruebe que se han instalado los controladores actualizados de la GPU.
- Compruebe que el servidor de licencias contiene suficientes licencias de DataHUB y la API Spectral.
- Compruebe el acceso a la red del servidor de licencias.
- La versión de Microsoft Visual Studio instalada debe ser compatible con .NET Framework 4.7.2.

6.3.2 Instalación de DataHUB para plug-in

El servicio DataHUB debe estar instalado en el PC de instalación. Durante la instalación, tiene la opción de especificar las licencias que desea. Seleccione la opción “LaserVIEW/MeltVIEW Pluggability”:

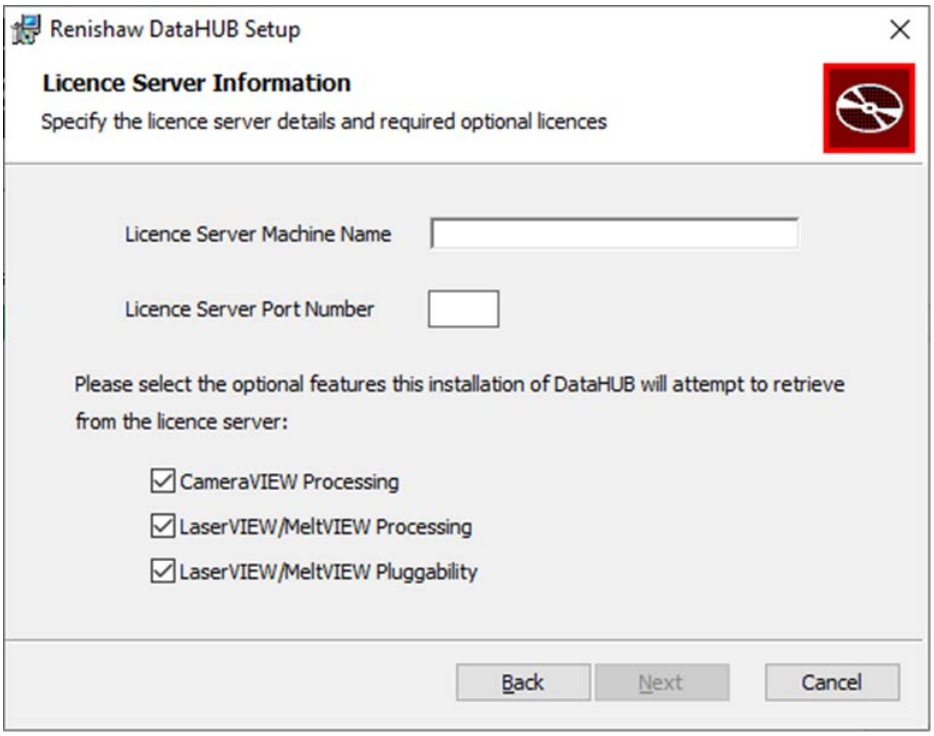


Imagen 6 Activación de licencia de DataHUB

6.3.3 Instalación del plug-in

En las siguientes secciones de este documento se explica el proceso completo de creación, compilación, depuración de errores e instalación de un plug-in de Visual Studio.

Los plug-in se programan en C# y definen una instalación de una clase pública de un juego específico de métodos públicos que DataHUB utiliza para identificar, validar y usar instancias del plug-in al procesar datos en el control de proceso de la construcción.

NOTA: Las capturas de pantalla siguientes están tomadas en Microsoft Visual Studio 2019, no obstante, el proceso de trabajo es muy similar en las demás versiones.

6.3.3.1 Creación de la solución Visual Studio

Inicie Visual Studio y elija *Crear un nuevo proyecto*. En las opciones, elija *Biblioteca de clases (.NET Framework)*:

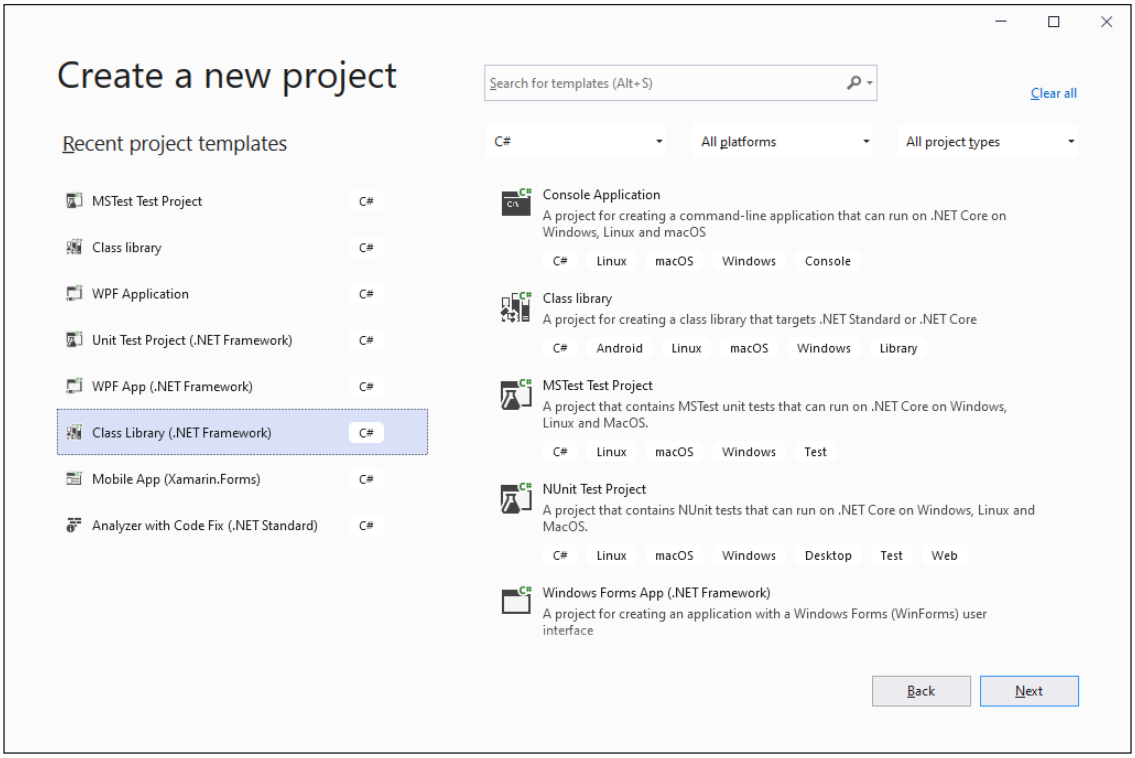


Imagen 7 Creación de la solución

A continuación, configure el proyecto:

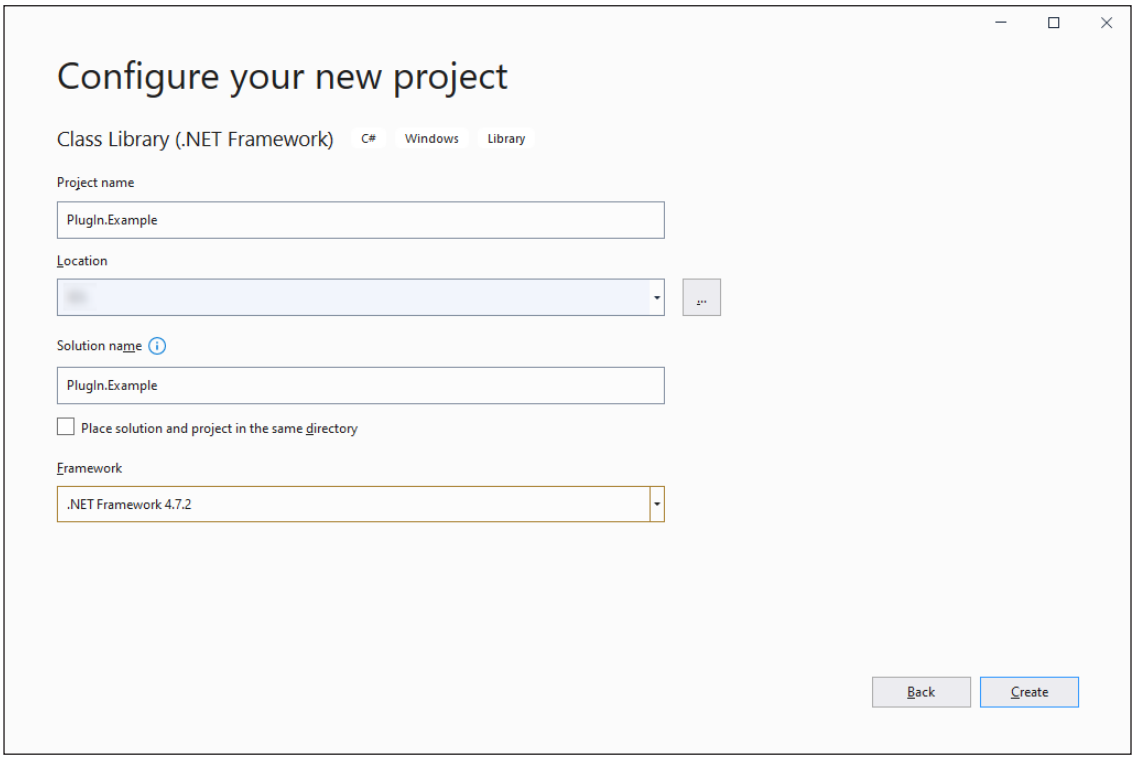


Imagen 8 Configuración del proyecto

DataHUB requiere un conjunto de plug-in para iniciar **PlugIn.** (La palabra “PlugIn” con el punto seguido). Seleccione una carpeta para la *Ubicación* del proyecto y cree la solución.

NOTA: DataHUB admite plug-in para x86/x64/Cualquier CPU. Se recomienda seleccionar Framework 4.7.2 o superior.

6.3.3.2 Asignación de nombre al tipo de plug-in

El archivo individual de origen por defecto Class1.cs contiene la definición de `Class1`, que se utiliza en este paso.

```
namespace PlugIn.Example
{
    public class Class1
    {
    }
}
```

Cuando DataHUB detecta un conjunto de plug-in posible, busca un `PlugIn` de tipo público, y cambia el nombre de `Class1` a `PlugIn`.

```
namespace PlugIn.Example
{
    public class Class1
    {
    }
}
```

6.3.3.3 Instalación de la API

Con un posible tipo de plug-in, DataHUB solicita que se instalen los siguientes métodos públicos:

`public PlugIn()` (generado automáticamente)
`public string APIVersion()`
`public string DisplayName()`
`public string Initialise(…)`
`public string ProcessPackets(…)`
`public string Shutdown()`

Añada los siguientes métodos de instalación a la clase:

```
public class Class1
{
   public string APIVersion() => “”;
   public string DisplayName() => “”;
   public string Initialise(
      string plugInFolderPath,
      string outputFolderPath,
      uint numberOfDatFilesPerLayer) => “”;
   public string ProcessPackets(
```

```
      uint layerNumber,
      uint laserId,
      int count,
      uint[] startTime,
      uint[] duration,
      double[] demandPositionX,
      double[] demandPositionY,
      uint[] laserVIEW,
      uint[] meltVIEWPlasma,
      uint[] meltVIEWMeltpool) => "";
   public string Shutdown() => "";
}
```

Todos los métodos de API vuelven a `string` (que se han dejado vacíos por el momento). El contenido de la cadena devuelta se utiliza para notificar a DataHUB el resultado del método. A medida que se describe cada método a continuación, se explica la función de este valor devuelto.

6.3.3.4 Método APIVersion

Este método notifica a DataHUB la versión de la API que utiliza el plug-in. Para usar la API versión 1.0, esta debe devolver el literal "1.0":

```
public string APIVersion() => "1.0";
```

6.3.3.5 Método DisplayName

El contenido de la cadena devuelta por este método se utiliza al cambiar el nombre mostrado del plug-in en DataHUB Monitor y al crear la carpeta de salida del plug-in. No es obligatorio que este nombre sea exclusivo, pero si lo es, facilita la identificación de plug-in, por ejemplo, cuando hay varias versiones instaladas en un DCPC.

```
public string DisplayName() => "Example plug-in";
```

6.3.3.6 Método Initialise

DataHUB invoca este método al iniciar una construcción, antes de enviar datos de capas al plug-in. Por tanto, pasa los datos de la *construcción sin variar* al plug-in. En este plug-in de ejemplo, el método *Initialise* asigna la ruta a la carpeta de salida de la instancia al campo **_outputFolderPath**:

```
public string Initialise(
   string plugInFolderPath,
   string outputFolderPath,
   uint numberOfDatFilesPerLayer)
{
   _outputFolderPath = outputFolderPath;
   return “Success”;
}
…
private string _outputFolderPath;
```

6.3.3.7 Método ProcessPackets

DataHUB invoca este método cuando ha recibido todos los datos de una capa. En el plug-in de ejemplo, el método *ProcessPackets* genera en primer lugar una línea de títulos de columna separados por comas y, a continuación, añade línea a línea los valores del paquete de datos de control del proceso:

```
public string ProcessPackets(
   uint layerNumber,
   uint laserId,
   int count,
   uint[] startTime,
   uint[] duration,
   double[] demandPositionX,
   double[] demandPositionY,
   uint[] laserVIEW,
   uint[] meltVIEWPlasma,
   uint[] meltVIEWMeltpool)
{
   var builder = new StringBuilder().
      Append($”Start time / us\t”).
      Append($”Duration / us\t”).
      Append($”x position / mm\t”).
      Append($”y position / mm\t”).
      Append($”LaserVIEW\t”).
      Append($”MeltVIEW Plasma\t”).
      Append($”MeltVIEW Melt Pool”).
```

```
        Append(Environment.NewLine);
    for (var i = 0; i < count; ++i)
        builder.
            Append($"{startTime[i]}").
            Append($"\t").
            Append($"{duration[i]}").
            Append($" \t").
            Append($"{demandPositionX[i].ToString("F3", CultureInfo.CurrentCulture)}\t").
            Append($"{demandPositionY[i].ToString("F3", CultureInfo.CurrentCulture)}\t").
            Append($"{laserVIEW[i]}\t").
            Append($"{meltVIEWPlasma[i]}\t").
            Append($"{meltVIEWMeltpool[i]}").
            Append(Environment.NewLine);
    System.IO.File.WriteAllText(
        Path.Combine(_outputFolderPath,
                    $"Packet data for layer {layerNumber}, laser {laserId}.txt"),
        builder.ToString());
    return "Success";
}
```

6.3.3.8 Método Shutdown

DataHUB invoca este método cuando se han enviado todos los datos de la construcción al plug-in o cuando han fallado las llamadas anteriores al plug-in. El método no tiene parámetros.

En este método, el plug-in debe realizar el procesamiento final de los datos de construcción, dejar libres los recursos utilizados, etc. En el plug-in de ejemplo, el método *Shutdown* es:

```
public string Shutdown() => "Success";
```

6.3.4 Instalación del plug-in para depuración de errores

Genere la solución en *Debug*, abra un explorador de archivos y busque la carpeta de salida que contiene un archivo *.dll* con el mismo nombre que el asignado al proyecto, por ejemplo, *PlugIn.Example.dll*. (También puede haber archivos antiguos de símbolos de depuración de errores, etc.)

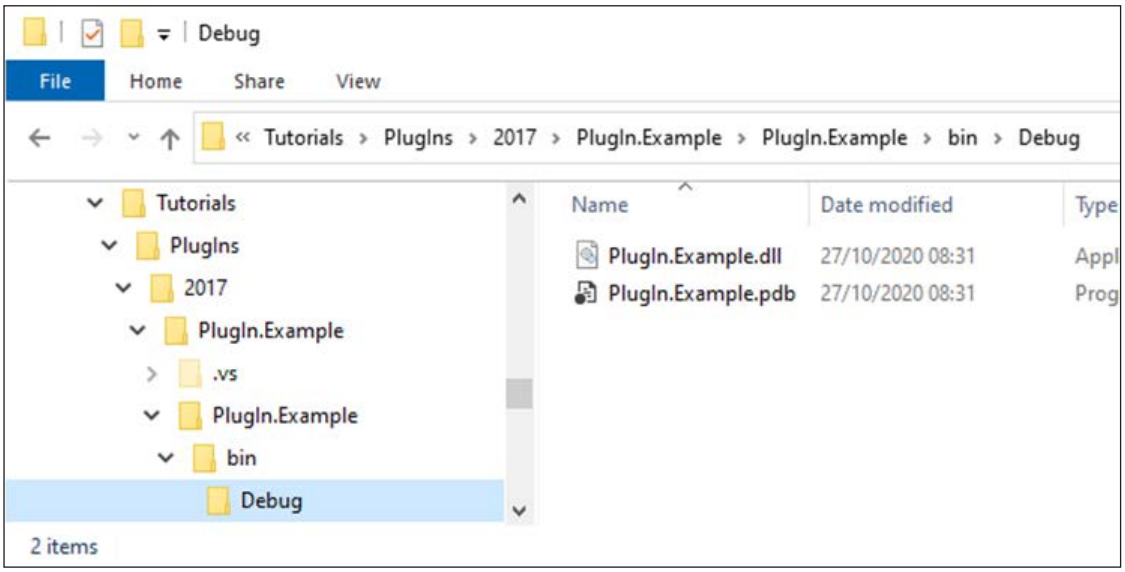


Imagen 9 Binarios de plug-in

Copie esta carpeta y péguela en la carpeta secundaria *PlugIns* de la carpeta de datos de DataHUB:

<$PROGRAMDATA$\Renishaw\DataHUB\PlugIns>.

NOTA: Normalmente, $PROGRAMDATA$ se encuentra <C:\ProgramData>, pero la carpeta puede estar oculta, en este caso, en el Explorador de archivos de Windows, seleccione la opción “Ver, Mostrar/ocultar, Elementos ocultos”.

Puede necesitar permiso de Administrador para modificar el contenido de esta carpeta.

Cambie el nombre de la carpeta *Debug* conforme al nombre del plug-in; en la imagen siguiente, el nombre de la carpeta es *PlugIn.Example*.

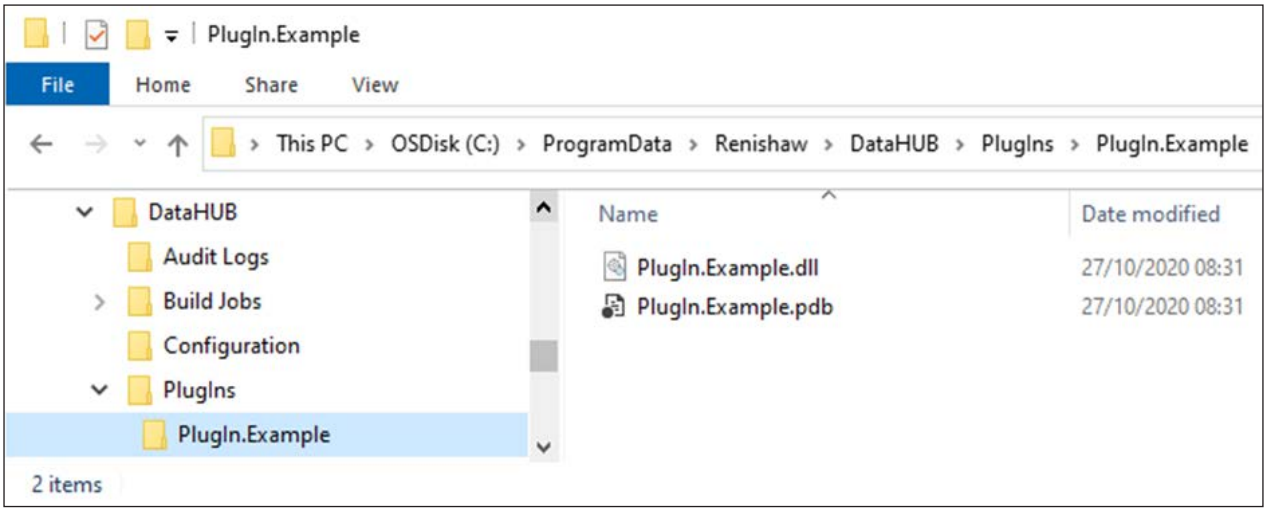


Imagen 10 Plug-in instalado

6.3.4.1 Registro del plug-in

Debe reiniciar el servicio DataHUB para registrar el nuevo plug-in. Para hacerlo, en la aplicación *Services*, haga clic con el botón derecho en *DataHUB Service* y seleccione *Reiniciar*.

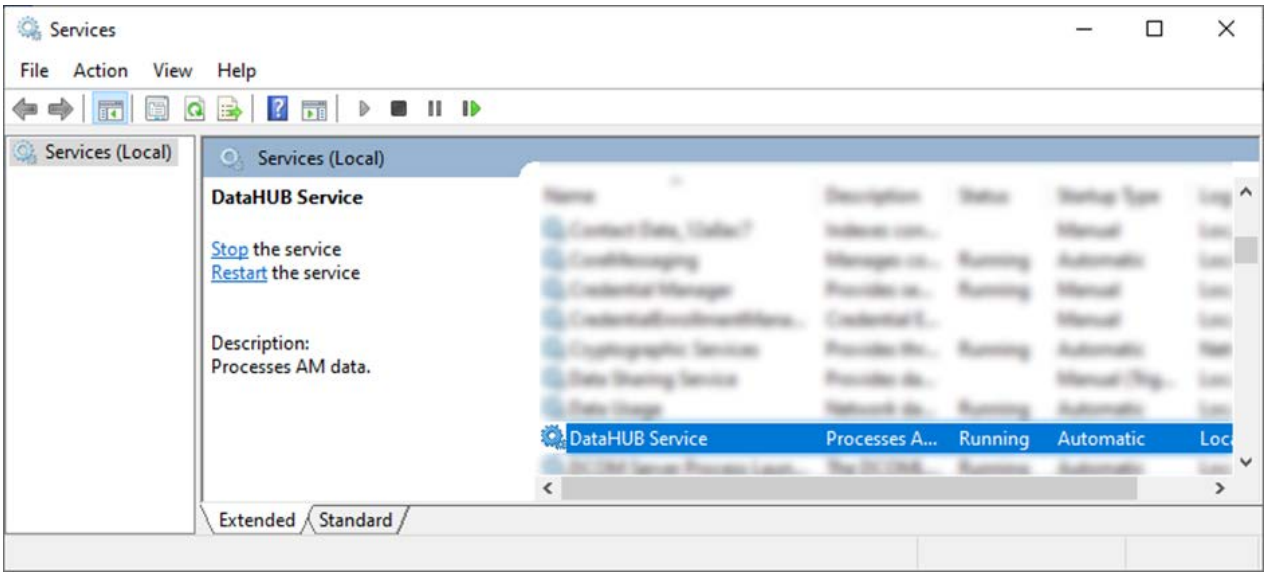


Imagen 11 Aplicación Windows Service

Pasados unos segundos, el servicio DataHUB muestra el estado como en ejecución *Running*.

6.3.4.2 Comprobación del registro

Durante su ejecución, DataHUB genera un archivo de registro que detalla los eventos importantes, como el inicio de las construcciones, los errores y, por supuesto, el control de plug-in. En el Explorador de archivos, busque la carpeta de registro <$PROGRAMDATA$\Renishaw\DataHUB\Audit Logs>.

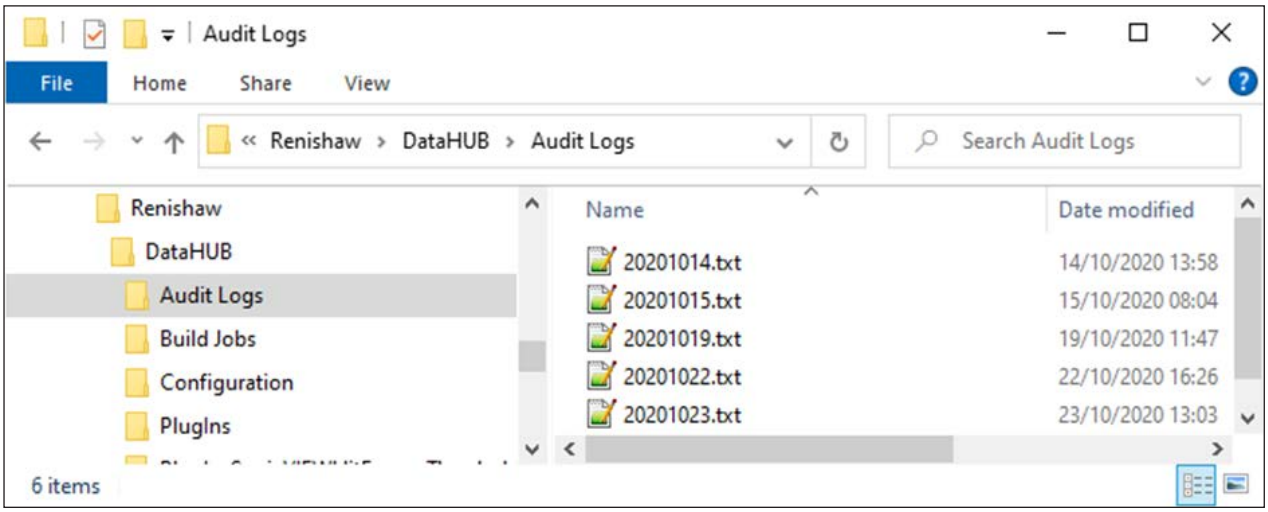


Imagen 12 Carpeta de registro de DataHUB

Abra el archivo de registro más reciente, el que tiene la fecha actual, vaya hasta el final y busque las líneas que detallan la localización de plug-in válidos:

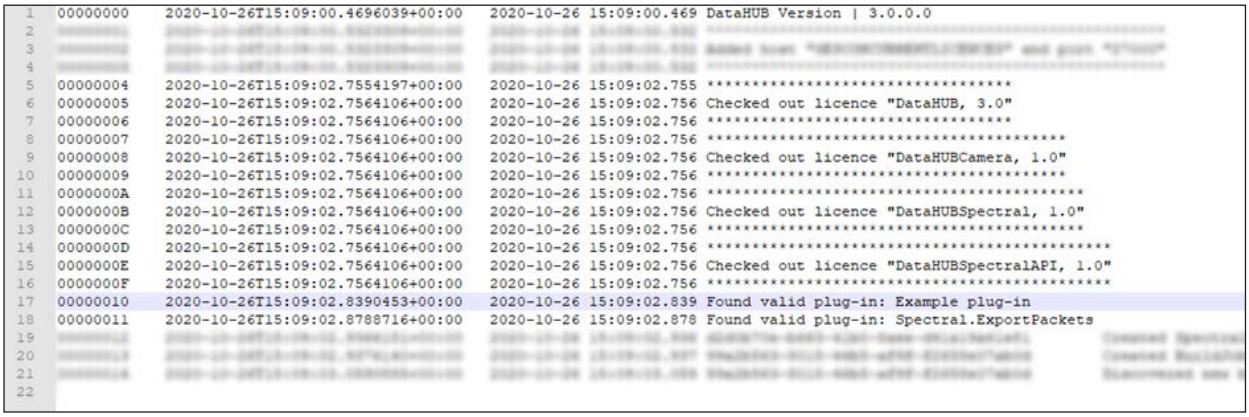


Imagen 13 Plug-in válidos detectados por DataHUB

6.3.5 Tiempo de espera

Puesto que DataHUB no puede garantizar que todas las llamadas a una instancia de plug-in se devuelvan a tiempo, aplica una política de tiempo de espera. Si un plug-in no se completa en un plazo de tiempo determinado, se presupone que es defectuoso y se desconecta. Los tiempos de espera son:

- Para *Initialise*, 20 segundos
- Para *ProcessPackets*, 200 segundos
- Para *Shutdown*, 20 segundos

Durante la depuración de errores, los tiempos de espera pueden sobrepasarse, por lo que DataHUB podría desconectar el plug-in prematuramente. Para disponer de más tiempo para depuración de errores, puede aumentar la duración en el archivo <$PROGRAMDATA$\Renishaw\DataHUB\PlugIns\PlugInsConfiguration.xml> en un editor de XML o texto.

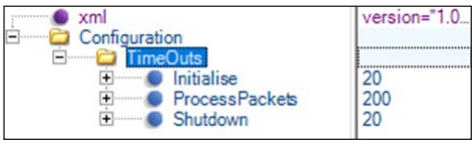


Imagen 14 Tiempos de espera del plug-in por defecto

NOTA: Es necesario reiniciar el servicio DataHUB para activar los cambios.

6.3.6 Depuración de errores del plug-in

Es posible depurar errores de ejecución en un plug-in correctamente instalado. El plug-in no se ejecuta directamente, pero, si conecta el programa de depuración de errores de Visual Studio al servicio DataHUB e inicia un trabajo de construcción, la instalación del plug-in se realiza a través del servicio y se controlan todos los puntos de parada definidos en el código.

NOTA: Para realizar la depuración de errores a través del servicio, Visual Studio puede necesitar permisos de Administrador. Para asignar estos derechos, haga clic con el botón derecho en el acceso directo del programa Visual Studio y elija “Ejecutar como administrador”.

En Visual Studio, defina puntos de parada al principio de los métodos *Initialise* y *ProcessPackets*:

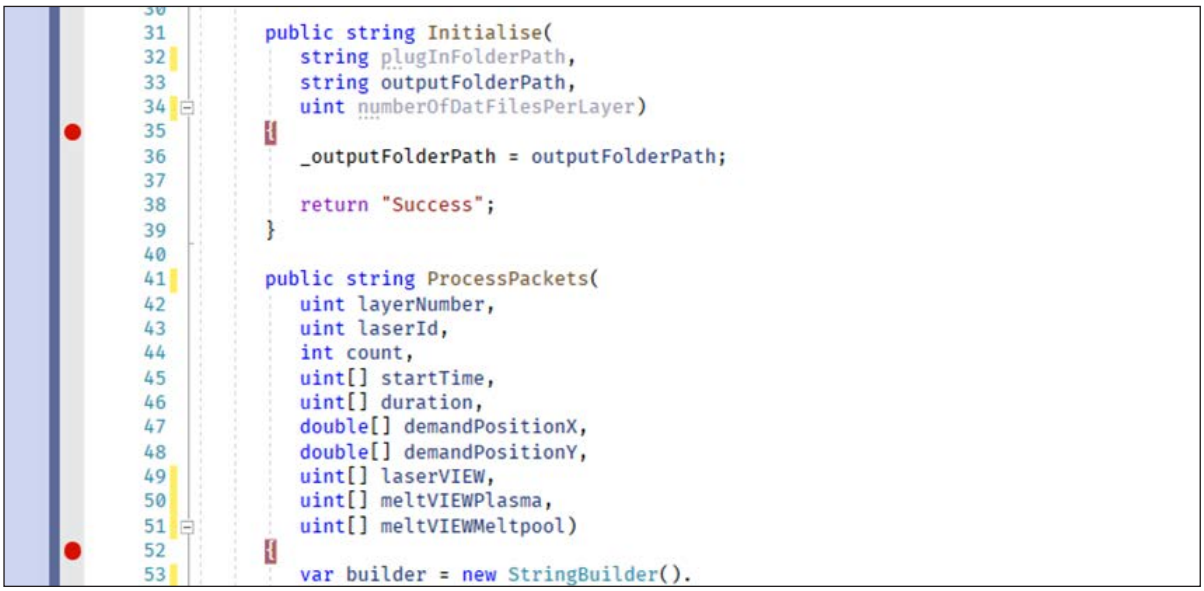


Imagen 15 Puntos de parada definidos en *Initialise* y *ProcessPackets*

Abra el menú Debug y seleccione “Adjuntar al proceso…”:

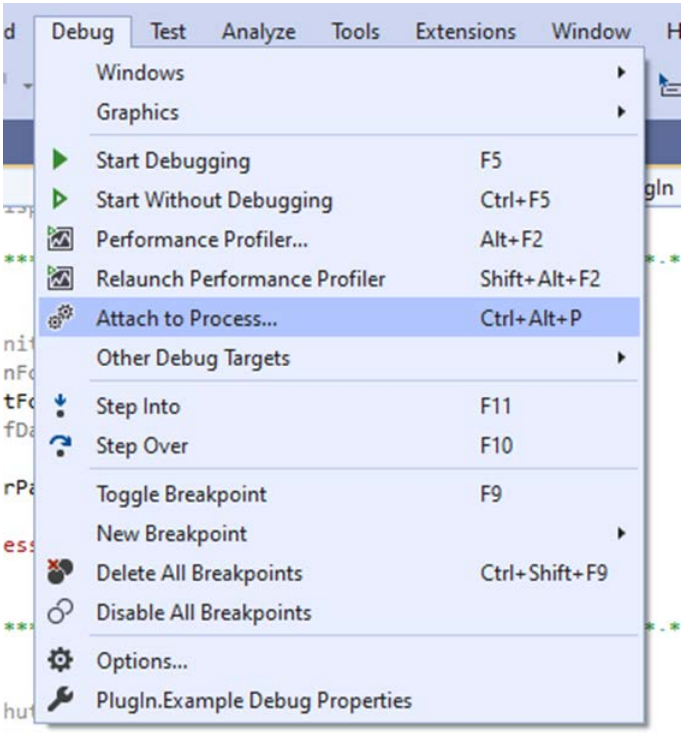


Imagen 16 Adjuntar a un proceso para depuración de errores

En la lista de procesos de ejecución, busque *DataHUB Service.exe*, si es necesario, seleccione “Mostrar procesos para todos los usuarios”:

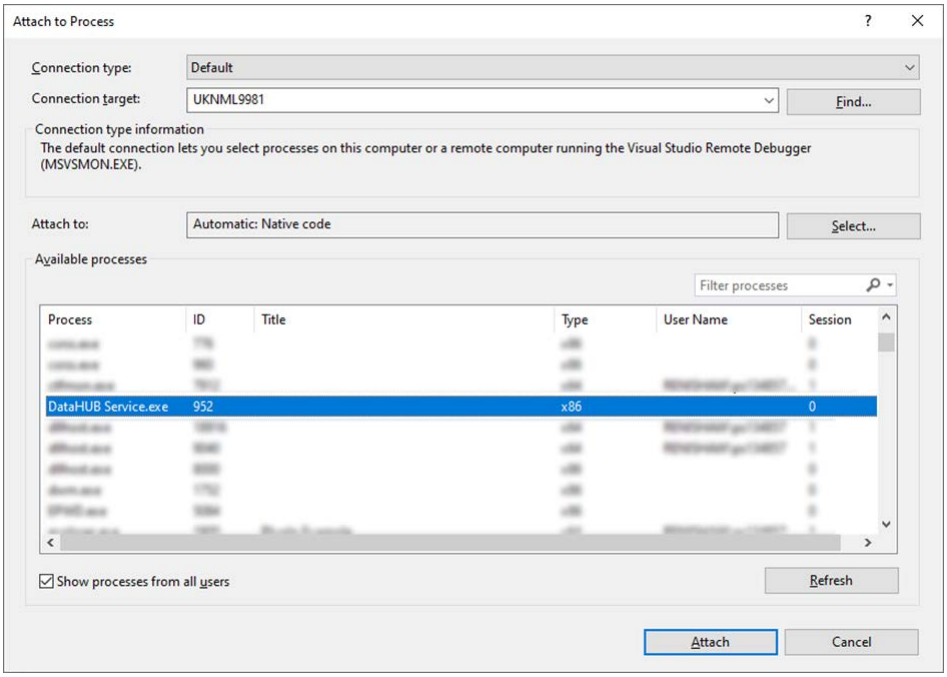


Imagen 17 Proceso del DataHUB Service

Al pulsar el botón “Asociar”, Visual Studio inicia una sesión de depuración de errores en el servicio DataHUB.

Para que DataHUB invoque el plug-in y los puntos de parada necesarios, debe iniciar una construcción. Puede hacerse en DataHUB Generator o, si ya ha generado otras construcciones, puede duplicar una carpeta de construcción en <$PROGRAMDATA$\Renishaw\DataHUB\Build Jobs>.

DataHUB detecta automáticamente la nueva construcción y empieza a procesar los datos. Una de las tareas preliminares consiste en crear (invocar constructor sin parámetros) una instancia de cada plug-in registrado. A continuación, cuando DataHUB invoca el método de la instancia *Initialise*, se alcanza el primer punto de parada. Los valores de los parámetros del método son los de la construcción, por ejemplo:

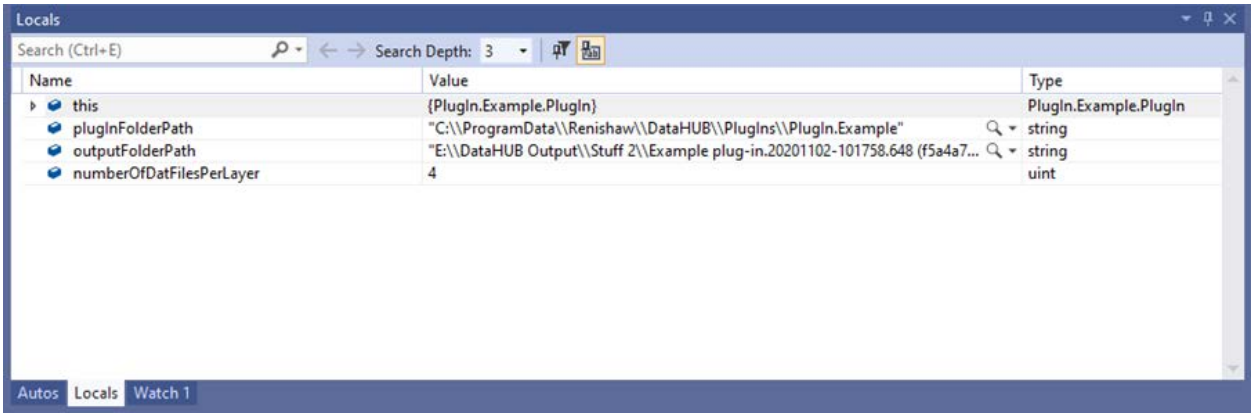


Imagen 18 Valores de parámetros del método *Initialise*

A continuación, se procesan los datos actuales (y, en una construcción activa, los datos a medida que se reciben) y se invoca el método *ProcessPackets* de la instancia con los valores de parámetros de los datos de LaserVIEW y MeltVIEW asociados a una capa:

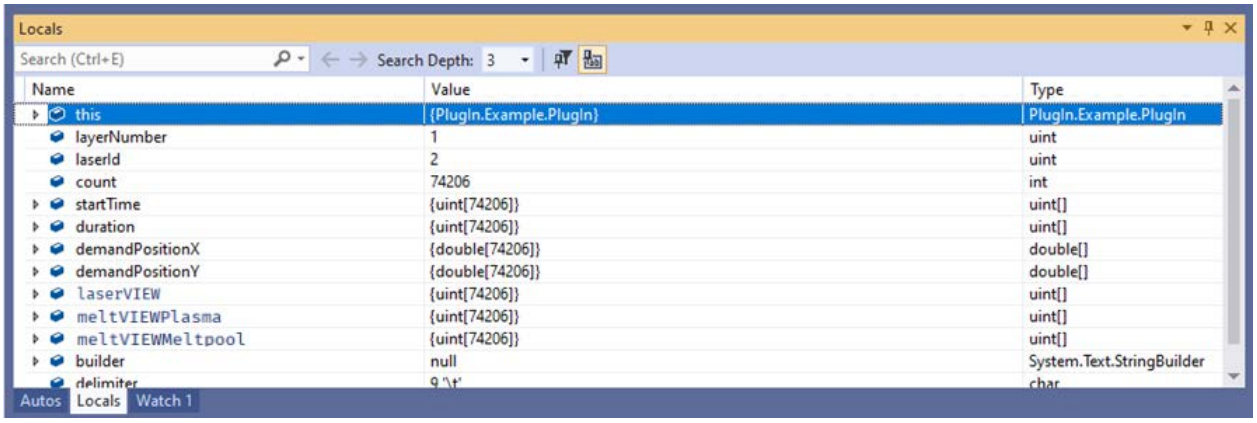


Imagen 19 Valores de parámetros del método *ProcessPacket*

El usuario puede elegir cómo añadir y controlar los puntos de parada en el método *Shutdown*.

6.3.7 Instalación del plug-in para producción

Para preparar la instalación del plug-in en un DCPC de producción, es imprescindible compilar una versión Release, distribuirla localmente y verificarla. Después de validad y verificar el plug-in, puede instalarlo en los DCPC para producción.

El proceso de instalación en un DCPC de producción es muy similar a la instalación en un PC de desarrollo. La carpeta *Release* debe copiarse en la carpeta secundaria *PlugIns* en la carpeta de datos de DataHUB (<$PROGRAMDATA$\Renishaw\DataHUB\PlugIns>) en el DCPC y cambiar el nombre del plug-in. A continuación, reinicie el servicio DataHUB para registrar el nuevo plug-in en la aplicación *Services*. El registro correcto o no del nuevo plug-in se anota en el registro de auditoría del día.

NOTA: DataHUB está diseñado para reanudar todas las tareas de procesamiento de datos en curso tras reiniciar el sistema, por lo que no es necesario esperar a que estas se completen. No obstante, solo las nuevas tareas de procesamiento de datos utilizan el plug-in registrado.

6.3.8 Diagnóstico de fallos del plug-in durante la producción

En el entorno de producción, DataHUB Monitor muestra (junto a otras tareas de procesamiento) el estado de procesamiento del plug-in:

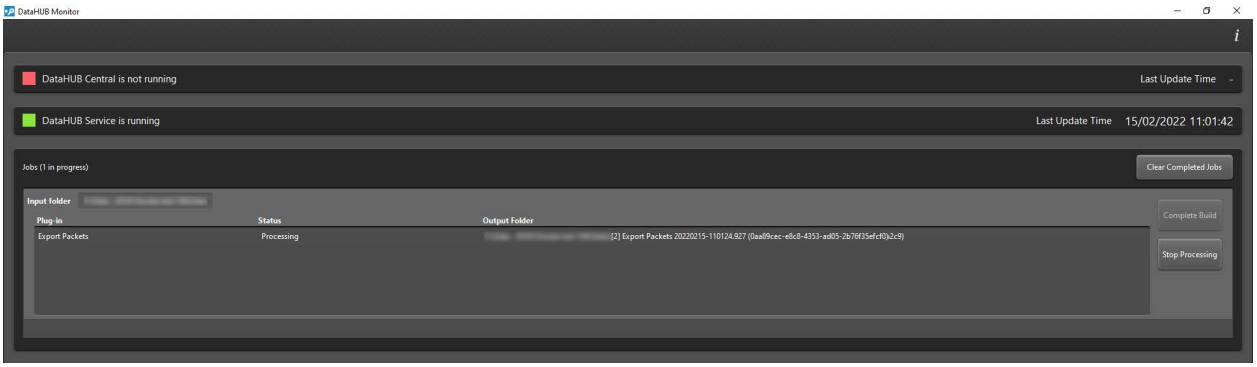


Imagen 20 Construcción en curso con plug-in

Si el estado de un plug-in es de *Error*, el registro de auditoría incluye los datos de este como entradas añadidas por DataHUB (por ejemplo, no se puede cargar el conjunto de plug-in debido a falta de dependencias) o en el plug-in a través de contenidos de valores de retorno de los métodos *Initialise*, *ProcessPackets* o *Shutdown*.

6.3.9 Localización de problemas

La tabla siguiente muestra fallos comunes, la entrada de registro de auditoría típica y posibles formas de solucionarlo:

Entrada del registro de auditoría	Error	Solución posible
El conjunto no contiene una clase denominada PlugIn	No se encuentra la clase PlugIn	Cambie el nombre de la clase del plug-in, repita la compilación y vuelva a instalarlo
La clase PlugIn no contiene un método denominado xyz	La clase PlugIn no instala uno de los métodos de API	Compruebe si existen errores en los nombres de los métodos
El método xyz no contiene la firma esperada …	La clase PlugIn no instala correctamente uno de los métodos de API	Examine la firma del método y compruebe que los parámetros coinciden con la definición de API
La clase PlugIn utiliza una versión incompatible del plugin API "{version}". Esta versión de DataHUB solo admite la versión "1.0" del plugin API.	La clase PlugIn no devuelve una cadena de versión API válida	Compruebe que el método APIVersion devuelve “1.0”
No se puede crear la carpeta de salida del plug-in <path>	No se ha creado la carpeta de una instancia específica de un plug-in	Compruebe los derechos de acceso de lectura/escritura/ modificación/creación de <path>
Resultado de Initialisation: tiempo de espera agotado Resultado de Process: tiempo de espera agotado Resultado de Shutdown: tiempo de espera agotado	Una llamada a uno de estos métodos API tarda demasiado tiempo en responder: se presupone que el plug-in ha fallado	Reduzca los trabajos realizados en el método o aumente la duración del tiempo de espera en PlugInsConfiguration.xml
Excepción:	Ha ocurrido un error desconocido	Examine los mensajes de excepción registrados para averiguar el origen del error

6.3.10 Integración de Renishaw Central

Los resultados del análisis (series cronológicas de puntos de datos, alertas, etc.) de un plug-in pueden enviarse al servidor de Renishaw Central. *Plug-in API V2.0 y los tutoriales* (n.º de referencia Renishaw H-5800-6784) contiene una explicación completa sobre cómo estructurar el contenido de la cadena devuelta por los métodos *Initialise* y *ProcessPackets* del plug-in para realizar la integración, así como los detalles de las funciones de ayuda de *PlugIns.API.v2.dll* compatibles con V1.0 API.

6.3.11 Plug-in API V1.0

6.3.11.1 Asignación de nombres del conjunto

El nombre de archivo del conjunto .NET debe empezar por “PlugIn.” (“PlugIn” seguido de un punto) y tener la extensión “dll”.

6.3.11.2 Asignación de nombres de la clase de plug-in

El conjunto de plug-in debe definir una clase pública denominada “`PlugIn`”.

6.3.11.3 Métodos de la clase de plug-in

La clase de plug-in debe instalar los siguientes métodos públicos:

```
public PlugIn()
public string APIVersion()
public string DisplayName()
public string Initialise(…)
public string ProcessPackets(…)
public string Shutdown()
```

6.3.11.4 Constructor sin parámetros

El tipo debe incluir un constructor sin parámetros. Si no se han definido constructores sin parámetros, C# los genera automáticamente, por tanto, no es necesario definirlos expresamente.

NOTA: Un plug-in no necesita ningún recurso en este constructor.

Una instancia de plug-in construida no garantiza la llamada a *Shutdown*, por tanto, es posible que estos recursos no se eliminen a tiempo. *Initialise* es la ubicación adecuada para adquirir recursos.

6.3.11.5 Método `APIVersion`

Identifica la versión de API en la que se debe validar el plug-in y el formato de datos de control de procesos que debe gestionar.

Parámetros:

Ninguno.

Devuelve: `string`

Un plug-in que instala la API V1.0 debe devolver el literal “1.0”.

6.3.11.6 Método `DisplayName`

El contenido de la cadena devuelta por este método se utiliza al cambiar el nombre mostrado del plug-in en DataHUB Monitor y al crear la carpeta de salida del plug-in. No es obligatorio que este nombre sea exclusivo, pero si lo es, facilita la identificación de plug-in, por ejemplo, cuando hay varias versiones instaladas en un DCPC.

Parámetros:

Ninguno.

Devuelve: `string`

El literal para utilizar en varias secciones de DataHUB UX, registro, etc.

6.3.11.7 Método `Initialise`

DataHUB invoca este método al iniciar una construcción, antes de enviar datos de capas al plug-in, por tanto, recibe los valores del parámetro de la *construcción sin variar*. El plug-in puede utilizar este método para asignar los recursos necesarios durante la vida útil de la instancia, calcular constantes de la construcción, etc.

Parámetro 1: `string`

Ruta de la carpeta del conjunto de plug-in, p.ej., <$PROGRAMDATA$\Renishaw\DataHUB\PlugIns\PlugIn.Example>

Parámetro 2: `string`

Ruta de la carpeta en la que este plug-in guarda los archivos de salida. Por cada plug-in y construcción, esta carpeta debe ser exclusiva e incluir el nombre del plug-in (como se define en el valor devuelto por el método `DisplayName`), una marca de fecha y GUID. Cada vez que se ejecuta el plug-in para una construcción, se cambia el nombre de la carpeta, pero siempre utiliza esta estructura:

<Carpeta de salida de la construcción>\[1.0] <Nombre del plug-in>.aaaaMMdd-HHmmss.fff (GUID)

Parámetro 3: `uint`

Es un valor del 1 a 4, como se define en la configuración de hardware de la máquina que genera los datos.

Devuelve: `string`

El plug-in debe devolver una cadena que contiene:

- la palabra “Success”, si el método se ha completado correctamente,
- un detalle de la causa del error o
- una cadena JSON conforme a la API devuelta a Renishaw Central.

Si DataHUB recibe una cadena no conforme, se desconecta el plug-in y se añade el contenido de la cadena devuelta al registro de auditoría para su posterior diagnóstico.

6.3.11.8 Método `ProcessPackets`

DataHUB invoca este método cuando ha recibido todos los datos de una capa en el DCPC. Los parámetros de este método son:

Parámetro 1: `uint`

Número de la capa de construcción a la que hacen referencia los datos. La primera capa es 1.

Parámetro 2: `uint`

Número de identificación “1”, “2”, “3” o “4” del láser para el que se obtienen los datos.

Parámetro 3: `int`

Recuento del número de elementos en los grupos siguientes.

Parámetro 4: `uint[ ]`

Hora de inicio de cada paquete, en µs, relativa al inicio de la capa.

Parámetro 5: `uint[ ]`

Duración de cada paquete en µs.

Parámetro 6: `double[ ]`

Posición de demanda X de cada paquete en la mesa del polvo, en mm.

Parámetro 7: `double[ ]`

Posición de demanda Y de cada paquete en la mesa del polvo, en mm.

Parámetro 8: `uint[ ]`

Valor promedio de todas las muestras obtenidas por el sensor LaserVIEW en la duración de cada paquete.

Parámetro 9: `uint[ ]`

Valor promedio de todas las muestras obtenidas por el sensor Plasma MeltVIEW en la duración de cada paquete.

Parámetro 10: `uint[ ]`

Valor promedio de todas las muestras obtenidas por el sensor Meltpool MeltVIEW en la duración de cada paquete.

Devuelve: `string`

El plug-in debe devolver:

- una cadena la palabra “Success”, si el método se ha completado correctamente,
- una cadena con el máximo detalle de la causa del error o
- una cadena JSON conforme a la API devuelta a Renishaw Central.

6.3.11.9 Método `Shutdown`

DataHUB invoca este método cuando se han enviado todos los datos de la construcción al plug-in o cuando han fallado las llamadas de *Initialise* al plug-in. En este método, el plug-in debe realizar el procesamiento final de los datos de construcción, dejar libres los recursos utilizados, etc. El método no tiene parámetros.

Parámetros:

Ninguno.

Devuelve: `string`

El plug-in debe devolver una cadena que contiene:

- la palabra “Success”, si el método se ha completado correctamente,
- una cadena con el detalle de la causa del error.

Si DataHUB recibe una cadena no conforme, se añade el contenido de la cadena devuelta al registro de auditoría para su posterior diagnóstico.

6.4 Plug-in API V2

Esta sección es una guía para la construcción, ensayo y desarrollo de un componente de software de plug-in V2.0 para DataHUB. El ciclo de desarrollo del plug-in se inicia al instalar DataHUB en el PC de desarrollo y, a continuación, se crea en Visual Studio un conjunto .NET que contiene el código del plug-in. Si Visual Studio está configurado correctamente para la depuración de errores, puede probar y corregir errores del plug-in para verificar si los valores son correctos antes de la instalación final en un PC de recopilación de datos (DCPC) en una máquina de FA de producción.

En esta sección, se define la estructura de una *secuencia de código* de una construcción. Un ejemplo de código explica cómo escribir una secuencia para procesar el plug-in en API V2.0. El segundo ejemplo de código muestra cómo usar los *filtros* para procesar una secuencia de código. A continuación, se representa la instalación del plug-in *ExportPackets* (distribuido con el software DataHUB). Los últimos ejemplos de código muestran cómo se pueden enviar los resultados del plug-in a Renishaw Central para mostrarlos junto a otros datos obtenidos en los sistemas de fabricación aditiva de Renishaw.

Esta sección se completa con una definición de API V2.0.

6.4.1 Actividades previas necesarias

Además de este documento, debe consultar los siguientes:

- Guía de instalación del software InfiniAM® y DataHUB (n.º de referencia Renishaw H-5800-6846)
- Guía de usuario de DataHUB (n.º de referencia Renishaw H-5800-6854)
- Manual de usuario del Administrador de licencias de Renishaw v2.x

Antes de continuar, debe realizar las siguientes tareas en el PC en el que se van a instalar los plug-in:

- El PC debe disponer de una GPU que cumpla como mínimo la especificación mínima de DataHUB (consulte la especificación de hardware del PC de recopilación de datos en la guía de instalación del software InfiniAM y DataHUB).
- Compruebe que se han instalado los controladores actualizados de la GPU.
- Compruebe que el servidor de licencias contiene suficientes licencias de DataHUB y la API Spectral.
- Compruebe el acceso a la red del servidor de licencias.
- La versión de Microsoft Visual Studio instalada debe ser compatible con .NET Framework 4.7.2.

6.4.2 Secuencias de código

6.4.2.1 Estructura de secuencia de código

El servicio DataHUB transforma los datos de control del proceso de una construcción en secuencias de código que utilizan los plug-in V2.0. El orden y el tipo de código de una secuencia crea una descripción seudo-jerárquica de los datos de control del proceso.

El código de una secuencia se divide en dos categorías: secuencias de código de inicio y final emparejadas, y código agrupad de inicio y final.

6.4.2.2 Secuencia de un láser para una capa

Los datos de control del proceso de un láser para una capa de una construcción se representan en la siguiente secuencia:

```
StartOfLayerLaserData (Layer, Laser)
   DataSourceInformation
   Sample
   Sample
   …
   Sample
EndOfLayerLaserData (Layer, Laser)
```

Observe que las secuencias de código emparejadas `StartOfLayerLaserData` y `EndOfLayerLaserData` crean una limitación conceptual alrededor del código específico del láser.

El código `DataSourceInformation` contiene valores de los orígenes de datos de control del proceso (p.ej., archivos DAT) que contienen los datos de ejemplo.

Cada código `Sample` contiene los valores de datos de control del proceso obtenidos en el hardware LaserVIEW y MeltVIEW para un ejemplo individual de 100 KHz.

6.4.2.3 Secuencia para una capa

Todos los datos de control del proceso de una capa se combinan en la secuencia de código, repitiendo la estructura por láser anterior una vez por cada láser de la máquina:

```
StartOfLayerData (Layer)
   StartOfLayerLaserData (Layer, Laser)
   …
   EndOfLayerLaserData (Layer, Laser)
   StartOfLayerLaserData (Layer, Laser)
      …
   EndOfLayerLaserData (Layer, Laser)
   StartOfLayerLaserData (Layer, Laser)
      …
```

```
    EndOfLayerLaserData (Layer, Laser)
    StartOfLayerLaserData (Layer, Laser)
       …
    EndOfLayerLaserData (Layer, Laser)
EndOfLayerData (Layer)
```

De nuevo, el código específico de la capa se agrupa por las cadenas `StartOfLayerData` y `EndOfLayerData` emparejadas.

6.4.2.4 Secuencia para una construcción

Todos los datos de una construcción crean una secuencia de código que repite la estructura por capa anterior por cada capa:

```
StartOfLayerData (Layer)
   …
EndOfLayerData (Layer)
StartOfLayerData (Layer)
   …
EndOfLayerData (Layer)
StartOfLayerData (Layer)
   …
EndOfLayerData (Layer)
StartOfLayerData (Layer)
   …
EndOfLayerData (Layer)
```

6.4.2.5 Separaciones

En la estructura de código descrita se ha omitido un pequeño detalle: el código de separación "`Split`". El código de separación se añade a la secuencia para marcar los puntos en los que cambia el carácter de los datos. Se añaden como una señal genérica, además de otro código `StartOf/EndOf`… más específico. A veces, al procesar una secuencia de código, es más sencillo ejecutar el código genérico que marcadores específicos.

6.4.2.6 Definición completa de una secuencia de código de una construcción

La definición completa de una secuencia de código es la siguiente:

```
StartOfLayerData (Layer)
   StartOfLayerLaserData (Layer, Laser)
      DataSourceInformation
      Separadas
      Sample
      Sample
      …
      Sample
      Separadas
   EndOfLayerLaserData (Layer, Laser)
   Separadas
   …
EndOfLayerData (Layer)
StartOfLayerData (Layer)
   …
EndOfLayerData (Layer)
```

6.4.2.7 Añadir un tipo de código

La API define una clase básica de código del que se deriva el resto. Para añadir un nuevo tipo de código a una cadena, puede derivarlo de esta clase:

```
public class NewTokenType : Token
{
}
```

Puede consultar un catálogo completo de código de API en la Sección 6.4, “Plug-in API V2”.

6.4.3 Instalación de DataHUB para plug-in

El servicio DataHUB debe estar instalado en el PC de instalación. Durante la instalación, tiene la opción de especificar las licencias que desea. Seleccione la opción “LaserVIEW/MeltVIEW Pluggability”:

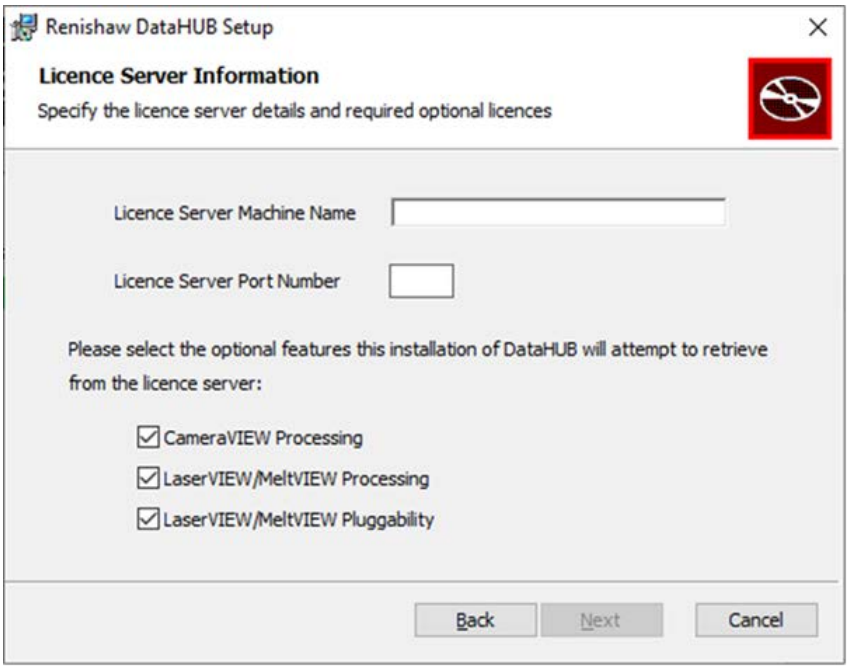


Imagen 21 Activación de licencia de DataHUB

6.4.4 Creación de un plug-in en Visual Studio

En las siguientes secciones de este documento se explica cómo crear una solución de plug-in en Visual Studio.

Los plug-in se programan en C# y definen una instalación de una clase pública de un juego específico de métodos públicos que DataHUB utiliza para identificar, validar y usar instancias del plug-in al procesar datos en el control de proceso de la construcción.

NOTA: En el proceso de trabajo siguiente se utiliza Microsoft Visual Studio 2019, no obstante, es muy similar a las demás versiones de Microsoft Visual Studio.

6.4.4.1 Creación de la solución

Inicie Visual Studio y elija *Crear un nuevo proyecto*. En las opciones, elija *Biblioteca de clases (.NET Framework)*:

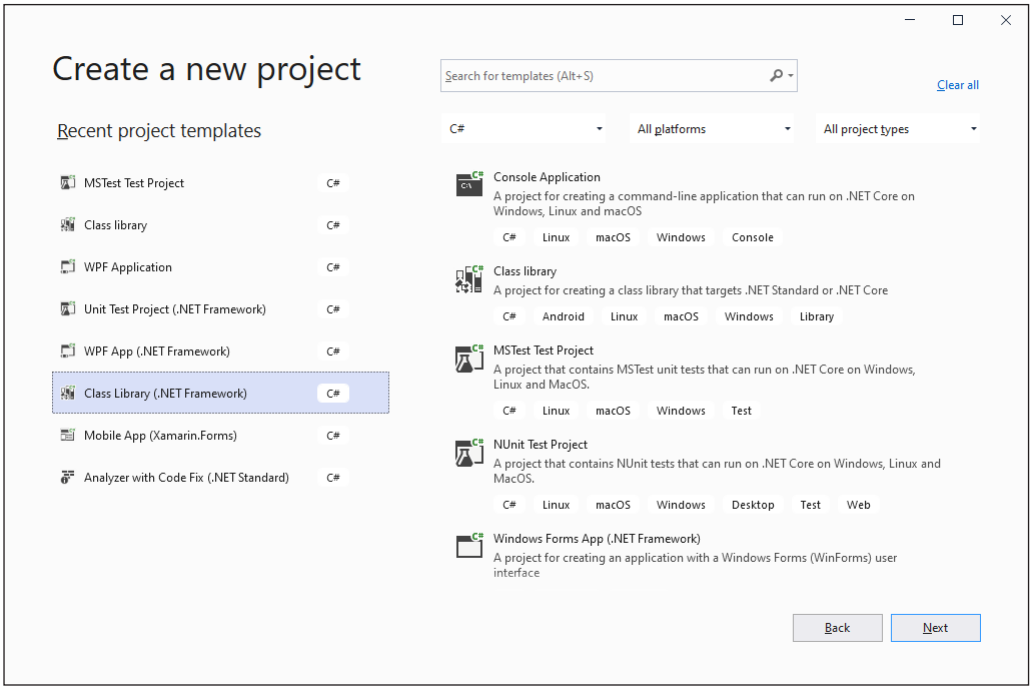


Imagen 22 Creación de la solución

A continuación, configure el proyecto:

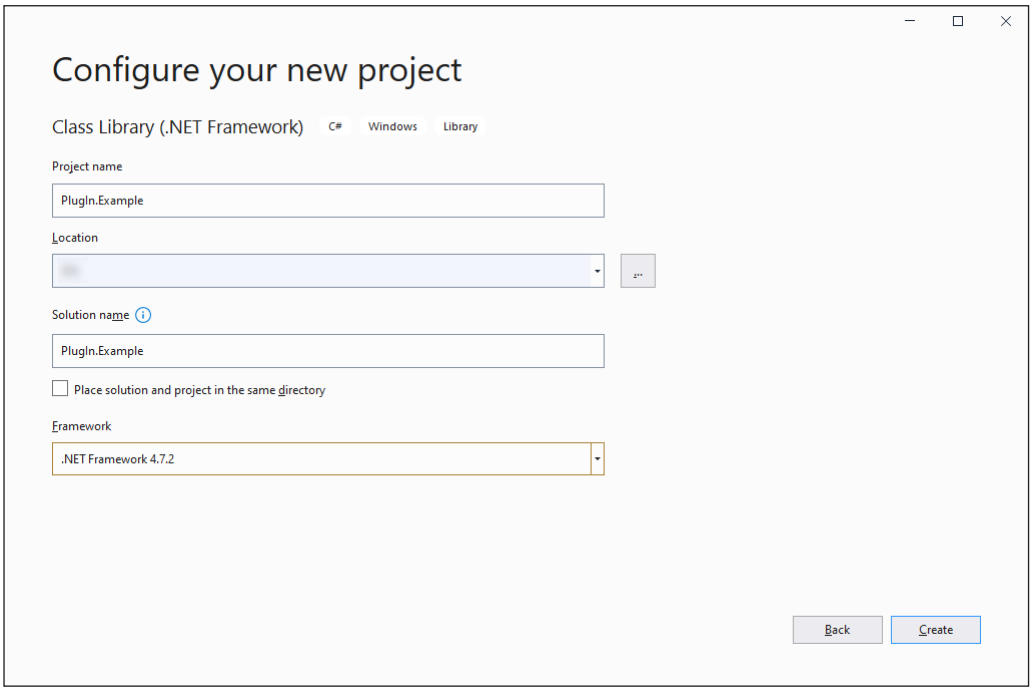


Imagen 23 Configuración del proyecto

DataHUB requiere un conjunto de plug-in para iniciar **PlugIn.** (La palabra “PlugIn” con el punto seguido). Seleccione una carpeta para la *Ubicación* del proyecto y cree la solución.

NOTA: DataHUB admite plug-in para x86/x64/Cualquier CPU. Se recomienda seleccionar Framework 4.7.2 o superior.

6.4.4.2 Añadir una referencia al conjunto de plug-in API

Para desarrollar un plug-in para V2.0 API, se necesita una referencia al conjunto de plug-in API. Haga clic con el botón derecho en el elemento de la solución “Referencias” y, a continuación, elija “Añadir referencia…”

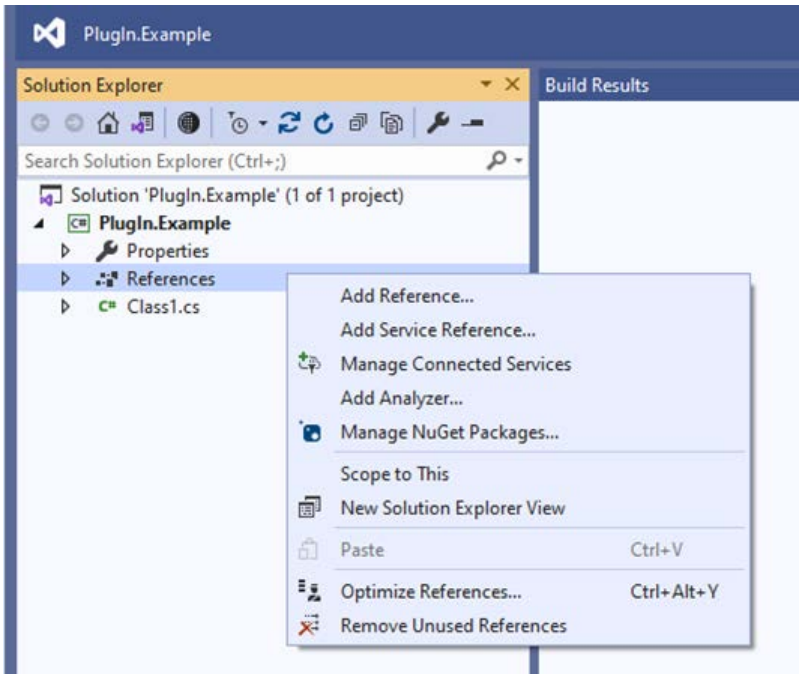


Imagen 24 Añadir referencia

En Reference Manager, elija “Examinar…” y busque el archivo “PlugIns.API.v2.dll” en la carpeta de instalación de DataHUB (es decir, <$PROGRAMFILES$\Renishaw\DataHUB>):

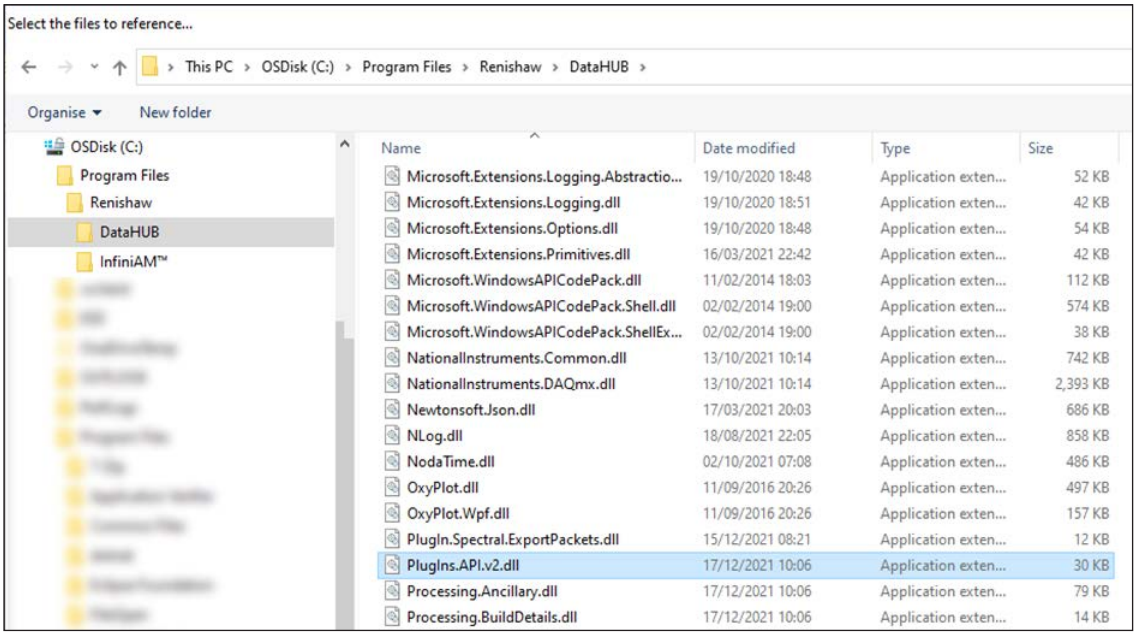


Imagen 25 Conjunto de API

El conjunto se muestra en las dependencias del proyecto de plug-in:

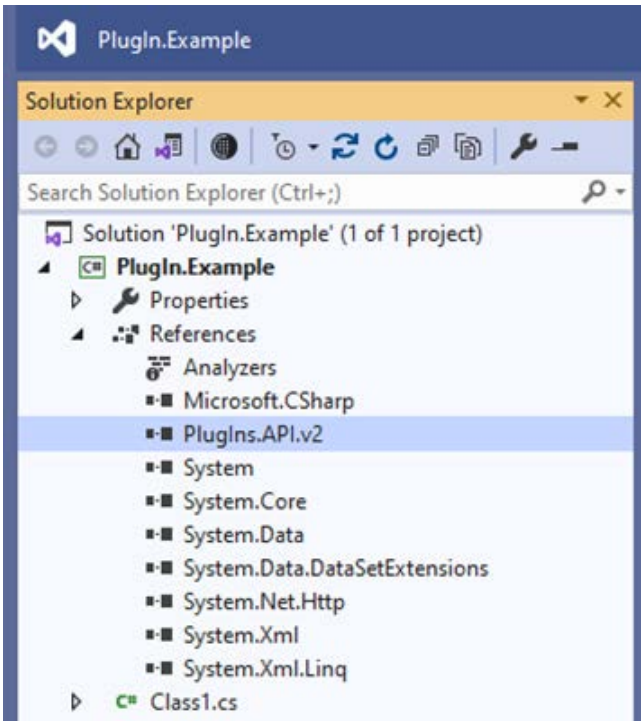


Imagen 26 La referencia al conjunto de plug-in API se realiza como dependencia

6.4.4.3 Asignación de nombre al tipo de plug-in

El archivo individual de origen por defecto Class1.cs contiene la definición de Class1, que se utiliza en este paso.

```
namespace PlugIn.Example
{
    public class Class1
    {
    }
}
```

Cuando DataHUB detecta un conjunto de plug-in posible, busca un `PlugIn` de tipo público, y cambia el nombre de `Class1` a `PlugIn`:

```
namespace PlugIn.Example
{
    public class Class1
    {
    }
}
```

6.4.4.4 Instalación de la API

Con un posible tipo de plug-in, DataHUB solicita que se instalen los siguientes métodos públicos:

```
public PlugIn() (generado automáticamente)
public static string APIVersion()
public static string DisplayName()
public string Initialise(…)
public string Process(…)
public string Shutdown()
```

Añada las siguientes declaraciones y métodos de instalación a la clase:

```
using System.Collections.Generic;
using PlugIns.API.v2.Helpers;
using PlugIns.API.v2.Types.Tokens;
public class PlugIn
{
   public static string APIVersion() => “2”;
   public static string DisplayName() => “Example plug-in”;
   public string Initialise(string initialisationData) => InitialiseReturns.Success();
   public string Process(IReadOnlyCollection<Token> tokens) => ProcessReturns.Success();
   public string Shutdown() => ShutdownReturns.Success();
}
```

Observe que todos los métodos API devuelven una cadena `string`. El contenido de la cadena devuelta se utiliza para notificar a DataHUB el resultado del método. A medida que se describe cada método siguiente, se explica la función de este valor devuelto.

6.4.4.5 Método `APIVersion` estático

Este método notifica a DataHUB la versión de la API que utiliza el plug-in. Para usar la API V2.0, esta debe devolver el literal “2”:

```
   public static string APIVersion() => “2”;
```

6.4.4.6 Método `DisplayName` estático

El contenido de la cadena devuelta por este método se utiliza al cambiar el nombre mostrado del plug-in en DataHUB Monitor, al crear la carpeta de salida del plug-in, etc. No es obligatorio que este nombre sea exclusivo, pero si lo es, facilita la identificación de Específico plug-in, por ejemplo, cuando hay varias versiones instaladas en un DCPC.

```
   public static string DisplayName() => “Example plug-in”;
```

6.4.4.7 Método `Initialise`

DataHUB invoca este método al iniciar una construcción, antes de enviar datos de capas al plug-in. Por tanto, pasa los datos de la *construcción sin variar* al plug-in. La instalación mínima debe devolver que el plug-in se ha inicializado sin fallos:

```
public string Initialise(string initialisationData) => InitialiseReturns.Success();
```

6.4.4.8 Método `Process`

DataHUB invoca este método cuando ha recibido todos los datos de una capa. La instalación mínima debe devolver que el plug-in se ha procesado sin fallos:

```
public string Process(IReadOnlyCollection<Token> tokens) => ProcessReturns.Success();
```

6.4.4.9 Método `Shutdown`

DataHUB invoca este método cuando se han enviado todos los datos de la construcción al plug-in o cuando han fallado las llamadas anteriores al plug-in. De nuevo, la instalación mínima no debe devolver fallos::

```
public string Shutdown() => ShutdownReturns.Success();
```

6.4.4.10 Instalación del plug-in para depuración de errores

Genere la solución en *Debug*, abra un explorador de archivos y busque la carpeta de salida que contiene un archivo *.dll* con el mismo nombre que el asignado al proyecto, por ejemplo, *PlugIn.Example.dll*. (También puede haber archivos antiguos de símbolos de depuración de errores, etc.)

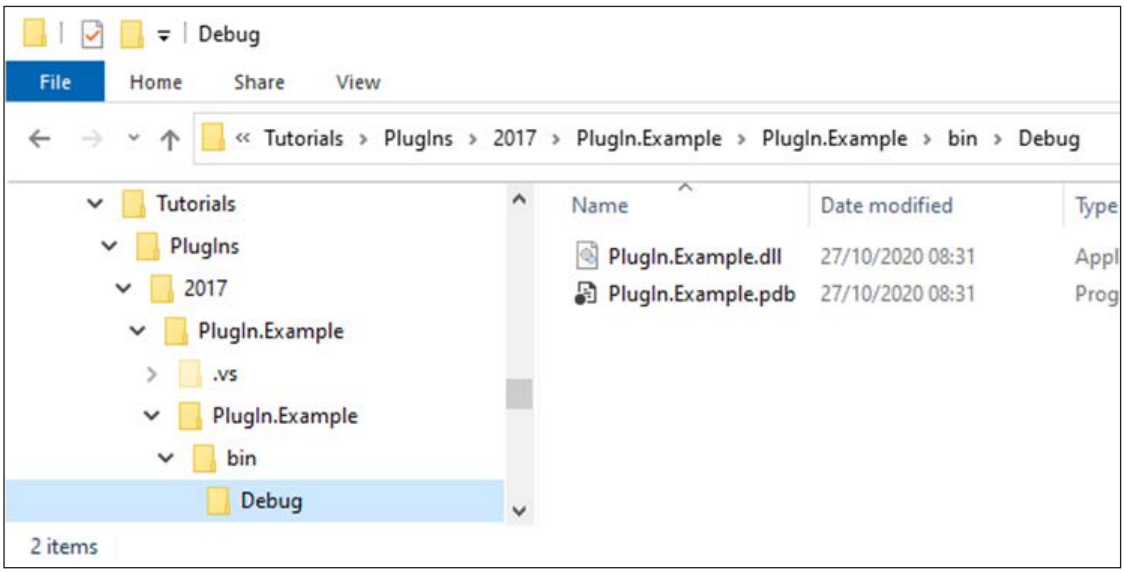


Imagen 27 Binarios de plug-in

Copie esta carpeta y péguela en la carpeta secundaria PlugIns de la carpeta de datos de DataHUB, p.ej., <$PROGRAMDATA$\Renishaw\DataHUB\PlugIns>.

NOTA: Normalmente, $PROGRAMDATA$ se encuentra <C:\ProgramData>, pero la carpeta puede estar oculta, en este caso, en el Explorador de archivos de Windows, seleccione la opción “Ver, Mostrar/ocultar, Elementos ocultos”.

Puede necesitar permiso de Administrador para modificar el contenido de esta carpeta.

Cambie el nombre de la carpeta *Debug* conforme al nombre del plug-in; en la imagen siguiente, el nombre de la carpeta es *PlugIn.Example*.

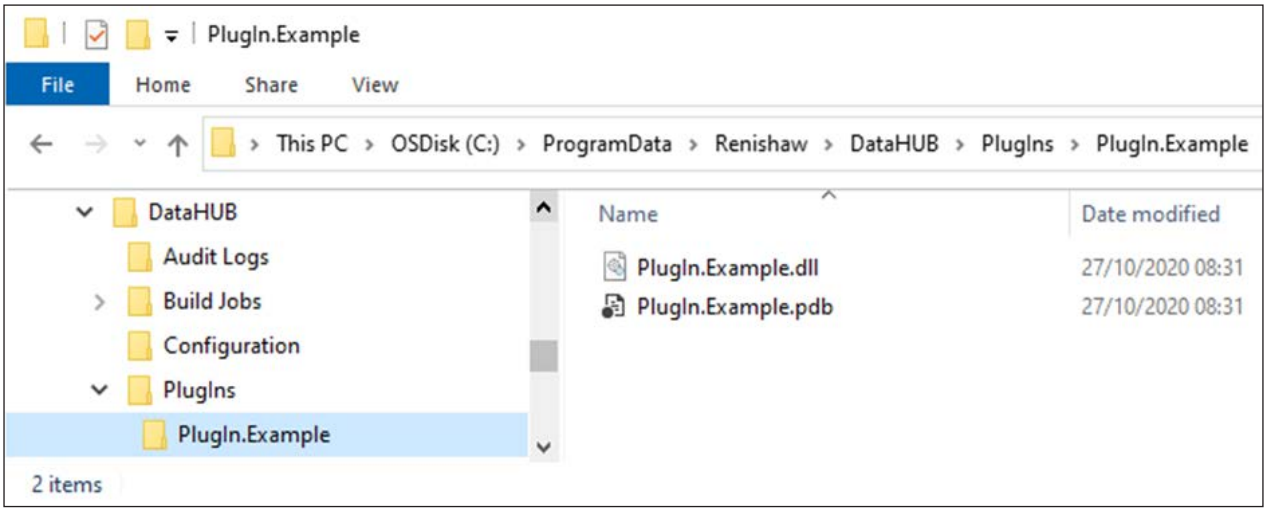


Imagen 28 Plug-in instalado

6.4.4.11 Registro del plug-in

Debe reiniciar el servicio DataHUB para registrar el nuevo plug-in. Para hacerlo, en la aplicación *Services*, haga clic con el botón derecho en *DataHUB Service* y seleccione *Reiniciar*.

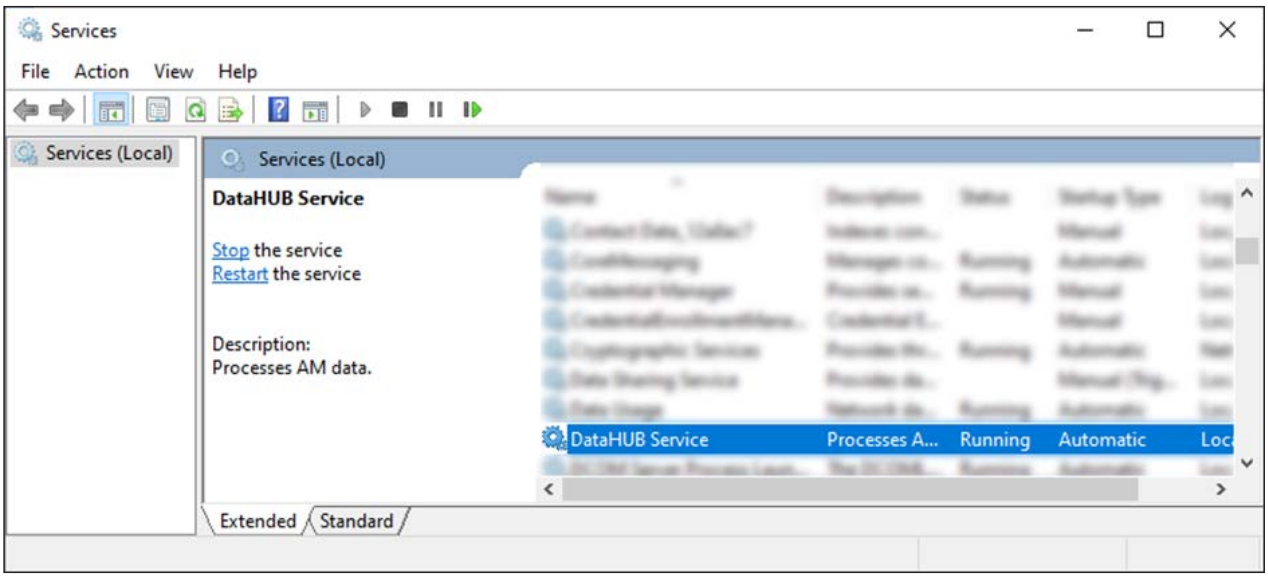


Imagen 29 Aplicación Windows Service

Pasados unos segundos, el servicio DataHUB muestra el estado como en ejecución *Running*.

6.4.4.12 Comprobación del registro

Durante su ejecución, DataHUB genera un archivo de registro que detalla los eventos importantes, como el inicio de las construcciones, los errores y, por supuesto, el control de plug-in. En el Explorador de archivos, busque la carpeta de registro <$PROGRAMDATA$\Renishaw\DataHUB\Audit Logs>:

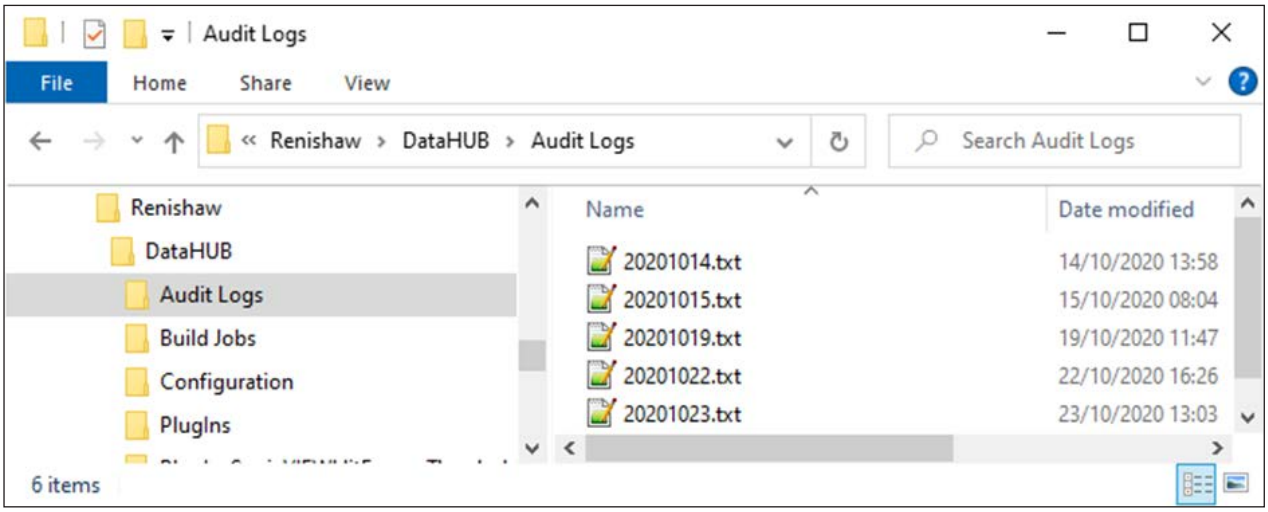


Imagen 30 Carpeta de registro de DataHUB

Abra el archivo de registro más reciente, el que tiene la fecha actual, vaya hasta el final y busque las líneas que detallan la localización de plug-in válidos:

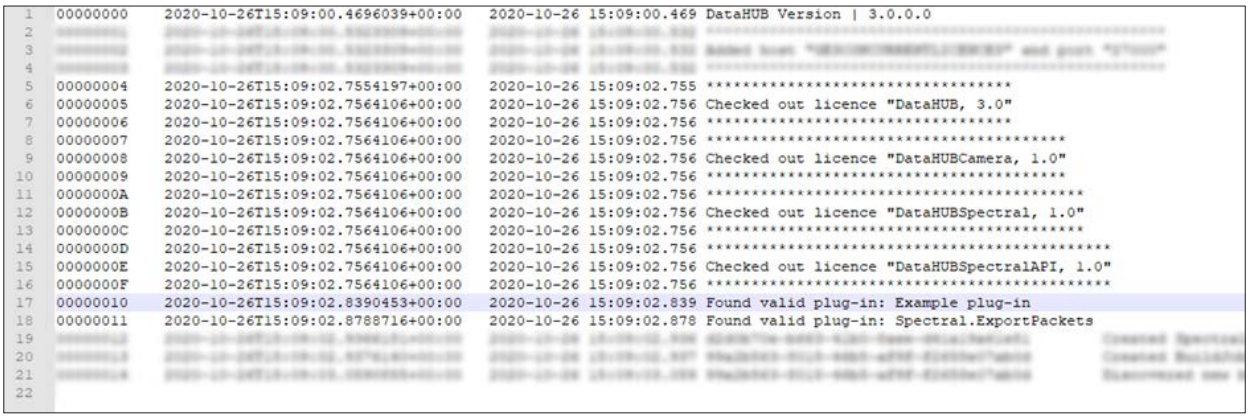


Imagen 31 Plug-in válidos detectados por DataHUB

6.4.4.13 Tiempos de espera

Puesto que DataHUB no puede garantizar que todas las llamadas a una instancia de plug-in se devuelvan a tiempo, aplica una política de tiempo de espera. Si un plug-in no se completa en un plazo de tiempo determinado, se presupone que es defectuoso y se desconecta. Los tiempos de espera son:

- Para *Initialise*, 20 segundos
- Para *Process* (compartido con API V1.0's *ProcessPackets*), 200 segundos
- Para *Shutdown*, 20 segundos

Durante la depuración de errores, los tiempos de espera pueden sobrepasarse, por lo que DataHUB podría desconectar el plug-in prematuramente. Para disponer de más tiempo para depuración de errores, puede aumentar la duración en el archivo <$PROGRAMDATA$\Renishaw\DataHUB\PlugIns\PlugInsConfiguration.xml> en un editor de XML o texto.

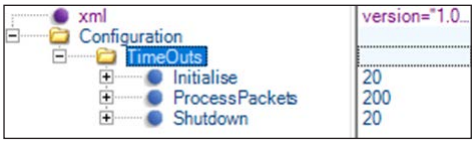


Imagen 32 Tiempos de espera del plug-in por defecto

NOTA: Es necesario reiniciar el servicio DataHUB para activar los cambios.

6.4.4.14 Depuración de errores del plug-in

Es posible depurar errores de ejecución en un plug-in correctamente instalado. El plug-in no se ejecuta directamente, pero, si conecta el programa de depuración de errores de Visual Studio al servicio DataHUB e inicia un trabajo de construcción, la instalación del plug-in se realiza a través del servicio y se controlan todos los puntos de parada definidos en el código.

NOTA: Para realizar la depuración de errores a través del servicio, Visual Studio puede necesitar permisos de Administrador. Para asignar estos derechos, haga clic con el botón derecho en el acceso directo del programa Visual Studio y elija "Ejecutar como administrador".

En Visual Studio, defina puntos de parada al principio de los métodos *Initialise*, *Process* y *Shutdown*:

Imagen 33 Puntos de parada definidos en *Initialise* y *ProcessPackets*

Abra el menú Debug y seleccione “Adjuntar al proceso…”:

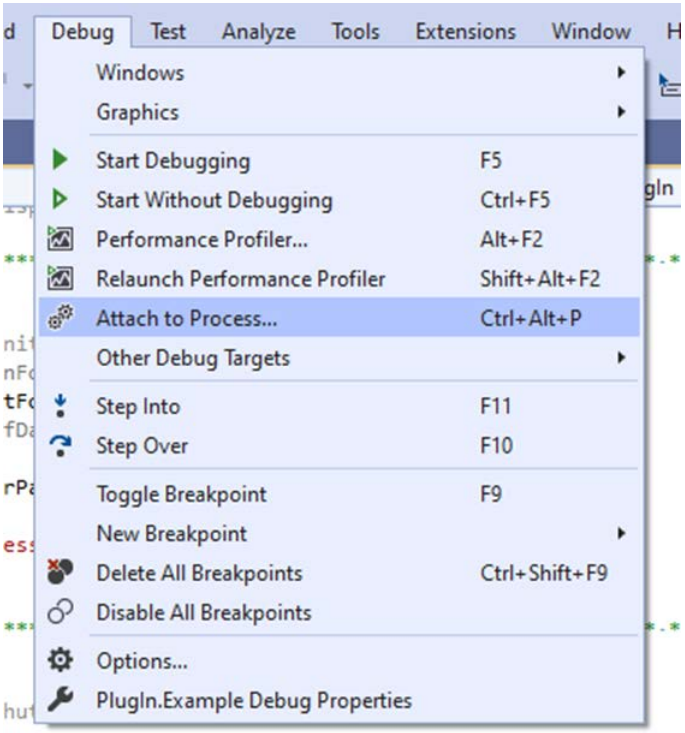


Imagen 34 Adjuntar a un proceso para depuración de errores

En la lista de procesos de ejecución, busque *DataHUB Service.exe*, si es necesario, seleccione “Mostrar procesos para todos los usuarios”:

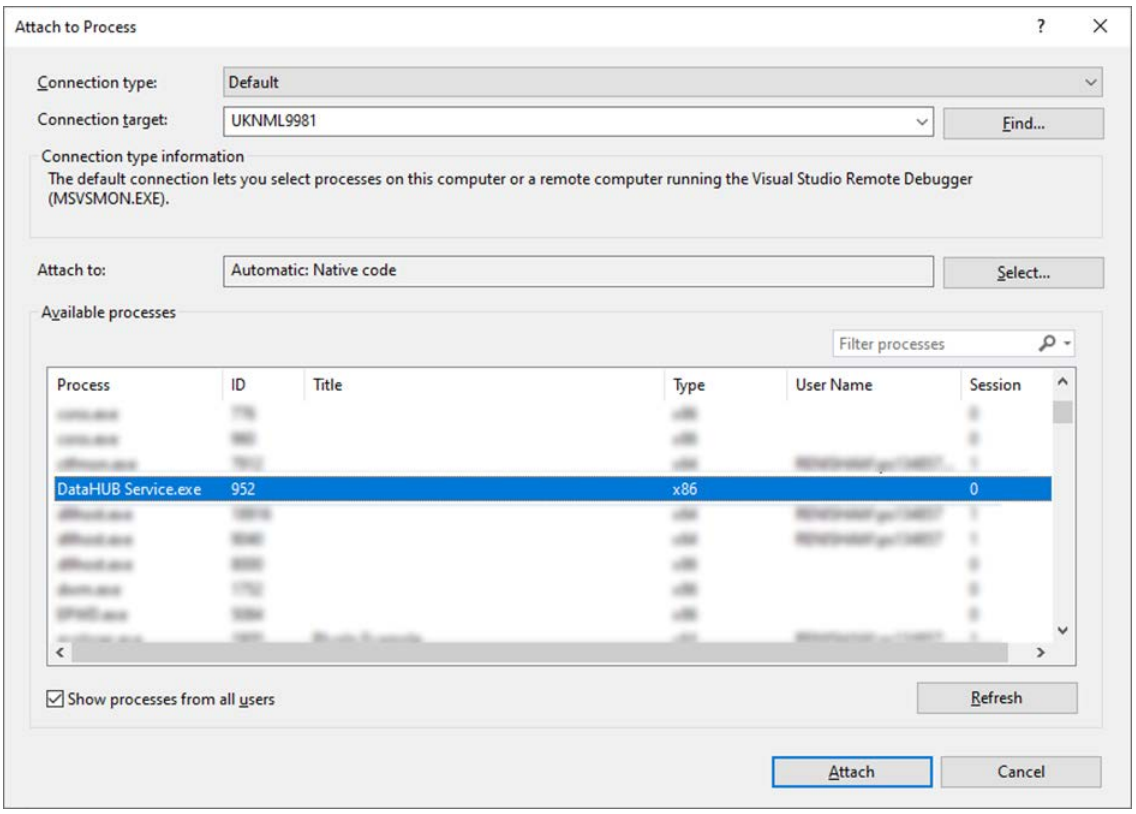


Imagen 35 Proceso del DataHUB Service

Al pulsar el botón “Asociar”, Visual Studio inicia una sesión de depuración de errores en el servicio DataHUB.

Para que DataHUB invoque el plug-in y los puntos de parada necesarios, debe iniciar una construcción. Puede hacerse en DataHUB Generator o, si ya ha generado otras construcciones, puede duplicar una carpeta de construcción en <$PROGRAMDATA$\Renishaw\DataHUB\Build Jobs>.

DataHUB detecta automáticamente la nueva construcción y empieza a procesar los datos. Una de las tareas preliminares consiste en crear (invocar constructor sin parámetros) una instancia de cada plug-in registrado. A continuación, cuando DataHUB invoca el método de la instancia *Initialise*, se alcanza el primer punto de parada. Los valores de los parámetros del método son los de la construcción, por ejemplo:

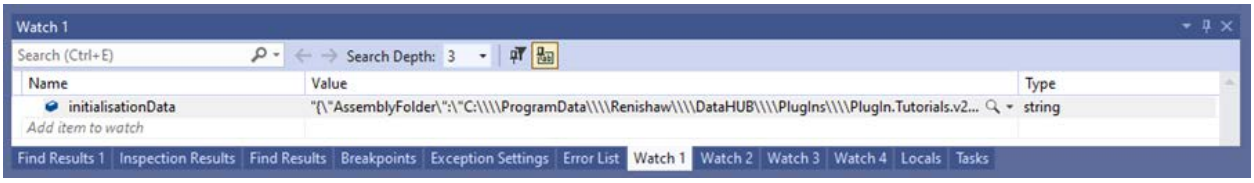


Imagen 36 Valores de parámetros del método Initialise

A continuación, se procesan los datos actuales (y, en una construcción activa, los datos a medida que se reciben) y se invoca el método *ProcessPackets* de la instancia con los valores de parámetros de los datos de LaserVIEW y MeltVIEW asociados a una capa:

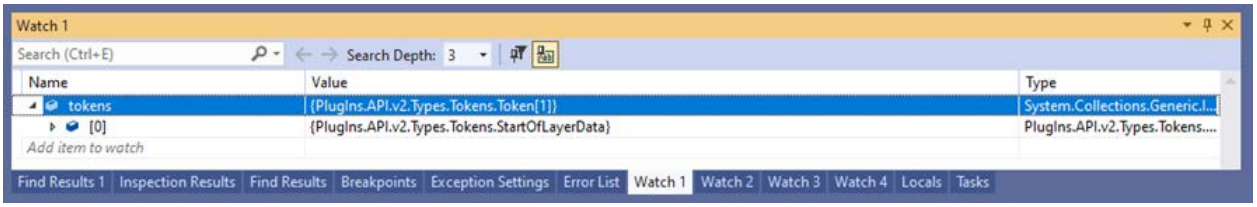


Imagen 37 Valores de parámetros del método Process

6.4.4.15 Instalación del plug-in para producción

Para preparar la instalación del plug-in en un DCPC de producción, es imprescindible compilar una versión *Release*, distribuirla localmente y verificarla. Después de validad y verificar el plug-in, puede instalarlo en los DCPC para producción.

El proceso de instalación en un DCPC de producción es muy similar a la instalación en un PC de desarrollo. La carpeta *Release* debe copiarse en la carpeta secundaria *PlugIns* en la carpeta de datos de DataHUB (<$PROGRAMDATA$\Renishaw\DataHUB\PlugIns>) en el DCPC y cambiar el nombre del plug-in. A continuación, reinicie el servicio DataHUB para registrar el nuevo plug-in en la aplicación *Services*. El registro correcto o no del nuevo plug-in se anota en el registro de auditoría .

NOTA: DataHUB está diseñado para reanudar todas las tareas de procesamiento de datos en curso tras reiniciar el sistema, por lo que no es necesario esperar a que estas se completen. No obstante, solo las nuevas tareas de procesamiento de datos utilizan el plug-in registrado.

6.4.4.16 Diagnóstico de fallos del plug-in durante la producción

En el entorno de producción, DataHUB Monitor muestra (junto a otras tareas de procesamiento) el estado de procesamiento del plug-in:

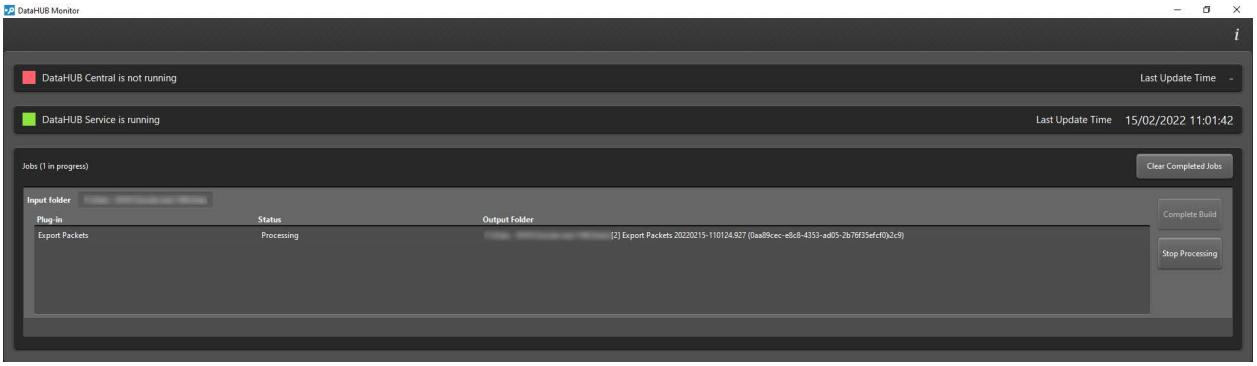


Imagen 38 Construcción en curso con plug-in

Si el estado de un plug-in es de *Error*, el registro de auditoría incluye los datos de este como entradas añadidas por DataHUB (por ejemplo, no se puede cargar el conjunto de plug-in debido a falta de dependencias) o en el plug-in a través de contenidos de valores de retorno de los métodos *Initialise*, *ProcessPackets* o *Shutdown*.

6.4.5 Ejemplo 1: Procesamiento de una cadena de código

Este primer ejemplo de código muestra cómo procesar una secuencia para extraer los valores máximo y mínimo de cada canal de datos de control del proceso, por cada láser y cada capa. Puede abarcar varios conceptos principales:

- Descodificación de datos de inicialización
- Procesamiento de código
- Carpeta de salida de la instancia

6.4.5.1 Creación

La instalación mínima del plug-in es la siguiente:

```
using System;
using System.Collections.Generic;
using System.IO;
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Helpers;
using PlugIns.API.v2.Types.Tokens;
namespace PlugIn.Tutorials.v2.Example1
{
   public class PlugIn
   {
      public static string APIVersion() => “2”;
      public static string DisplayName() => “Tutorial Example 1”;
      public string Initialise(string initialisationData) => InitialiseReturns.Success();
      public string Process(IreadOnlyCollection<Token> tokens) => ProcessReturns.Success();
      public string Shutdown() => ShutdownReturns.Success();
   }
}
```

Las clases de ayuda de API `InitialiseReturns`, `ProcessReturns` y `ShutdownReturns` proporcionan los métodos adecuados para generar los valores que devuelve API; los métodos correctos devuelven una cadena estándar “Success” que reconoce DataHUB.

6.4.5.2 Inicialización

Cuando DataHUB inicia este plug-in, la primera tarea que realiza es almacenar la ubicación de la carpeta de salida exclusiva de la instancia del plug-in para su uso posterior al grabar el archivo. Este método descodifica (mediante la biblioteca Newtonsoft.Json) la cadena `initialisationData` y extrae la ruta de la carpeta de salida:

```
public class PlugIn
{
   …
   public string Initialise(string initialisationData)
   {
      var initialisationValues = Jobject.Parse(initialisationData);
      _outputFolder = initialisationValues[“OutputFolder”].Value<string>();
      return InitialiseReturns.Success();
   }
   private string _outputFolder;
   …
}
```

NOTA: Los datos de inicialización se codifican como cadenas con formato JSON.

6.4.5.3 Procesamiento de secuencias de código

Si la inicialización es correcta, la instancia de plug-in recibe la secuencia de código de la construcción. Puesto que algunos juegos de datos de control de procesos pueden ser muy grandes, la secuencia de código se entrega por lotes; cada vez que se invoca `Process`, se recibe el siguiente lote de la secuencia de código.

A continuación, se puede añadir la gestión de código `Sample` al método `Process`:

```
…
public string Process(IreadOnlyCollection<Token> tokens)
{
   foreach (var token in tokens)
      switch (token)
      {
         case Sample sample:
            _demandX             = MinMaxForProperty(_demandX,             sample.DemandX);
            _demandY             = MinMaxForProperty(_demandY,             sample.DemandY);
            _demandFocus         = MinMaxForProperty(_demandFocus,         sample.DemandFocus);
            _demandLaserPower    = MinMaxForProperty(_demandLaserPower,    sample.DemandLaserPower);
            _laserModulate       = MinMaxForProperty(_laserModulate,       sample.LaserModulate);
            _meltVIEWPlasma      = MinMaxForProperty(_meltVIEWPlasma,      sample.MeltVIEWPlasma);
```

```
            _meltVIEWMeltpool    = MinMaxForProperty(_meltVIEWMeltpool,    sample.MeltVIEWMeltpool);
            _laserVIEW           = MinMaxForProperty(_laserVIEW,           sample.LaserVIEW);
            _laserOutputPower    = MinMaxForProperty(_laserOutputPower,    sample.LaserOutputPower);
            _laserBackReflection = MinMaxForProperty(_laserBackReflection, sample.LaserBackReflection);
            break;
      }
   return ProcessReturns.SuccessWithoutInformation();
}
private static (double, double) MinMaxForProperty((double Min, double Max) currentValue, double value)
{
   var (min, max) = currentValue;
   return (Math.Min(min, value), Math.Max(max, value));
}
private static (int, int) MinMaxForProperty((int Min, int Max) currentValue, int value)
{
   var (min, max) = currentValue;
   return (Math.Min(min, value), Math.Max(max, value));
}
private (double Min, double Max) _demandX             = (double.MaxValue, double.MinValue);
private (double Min, double Max) _demandY             = (double.MaxValue, double.MinValue);
private (double Min, double Max) _demandFocus         = (double.MaxValue, double.MinValue);
private (int Min, int Max)       _laserModulate       = (int.MaxValue, int.MinValue);
private (int Min, int Max)       _demandLaserPower    = (int.MaxValue, int.MinValue);
private (int Min, int Max)       _meltVIEWPlasma      = (int.MaxValue, int.MinValue);
private (int Min, int Max)       _meltVIEWMeltpool    = (int.MaxValue, int.MinValue);
private (int Min, int Max)       _laserVIEW           = (int.MaxValue, int.MinValue);
private (int Min, int Max)       _laserOutputPower    = (int.MaxValue, int.MinValue);
private (int Min, int Max)       _laserBackReflection = (int.MaxValue, int.MinValue);
…
```

Se inician varios campos, que almacenan la ejecución mínima y máxima de las propiedades de los datos de control de procesos de un código `Sample`.

Observe que el valor devuelto de `Process` ha cambiado a `SuccessWithoutInformation()`, que indica a DataHUB que el plug-in ha completado el procesamiento del lote de código, pero no quiere añadir un mensaje a los registros de DataHUB; este cambio asegura que los registros de DataHUB no se cargan con mensajes “Success” innecesarios.

NOTA: Las secuencias de código se entregan por lotes; `Process` se invoca varias veces.

NOTA: No todas las llamadas a `Process` tienen que devolver un mensaje de registro al servicio DataHUB.

El valor del código `Sample` de las propiedades de los datos de control de procesos se utiliza para actualizar la ejecución máxima y mínima. Algunas propiedades son de tipo doble y otras entero.

NOTAS: Un código `Sample` tiene las siguientes propiedades de datos de control de proceso:

- DemandX (doble)
- DemandY (doble)
- DemandFocus (doble)
- DemandLaserPower (entero)
- LaserModulate (entero)
- MeltVIEWPlasma (entero)
- MeltVIEWMeltpool (entero)
- LaserVIEW (entero)
- LaserOutputPower (entero)
- LaserBackReflection (entero)

6.4.5.4 Guardar en archivo

La tarea final del plug-in es guardar el resultado en un archivo después de procesar el código `EndOfLayerLaserData`:

```
…
public string Process(IreadOnlyCollection<Token> tokens)
{
   foreach (var token in tokens)
      switch (token)
      {
         …
         case EndOfLayerLaserData endOfLayerLaserData:
            var layer = endOfLayerLaserData.Layer.Value;
            var laser = endOfLayerLaserData.Laser.Value;
            var outputFilePath = Path.Combine(_outputFolder, “Min & Max for “
                                                          + $”layer {layer}, “
                                                          + $”laser {laser}”
                                                          + “.txt”);
            File.AppendAllText(outputFilePath, FormatMinAndMax(“DemandX”,            _demandX));
            File.AppendAllText(outputFilePath, FormatMinAndMax(“DemandY”,            _demandY));
            File.AppendAllText(outputFilePath, FormatMinAndMax(“DemandFocus”,        _demandFocus));
```

```
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandLaserPower",     _demandLaserPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserModulate",        _laserModulate));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWPlasma",       _meltVIEWPlasma));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWMeltpool",     _meltVIEWMeltpool));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserVIEW",            _laserVIEW));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserOutputPower",     _laserOutputPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserBackReflection", _laserBackReflection));
            _demandX              = (double.MaxValue, double.MinValue);
            _demandY              = (double.MaxValue, double.MinValue);
            _demandFocus          = (double.MaxValue, double.MinValue);
            _laserModulate        = (int.MaxValue, int.MinValue);
            _demandLaserPower     = (int.MaxValue, int.MinValue);
            _meltVIEWPlasma       = (int.MaxValue, int.MinValue);
            _meltVIEWMeltpool     = (int.MaxValue, int.MinValue);
            _laserVIEW            = (int.MaxValue, int.MinValue);
            _laserOutputPower     = (int.MaxValue, int.MinValue);
            _laserBackReflection  = (int.MaxValue, int.MinValue);
            return ProcessReturns.Success($"Processed layer {layer}, laser {laser}");
        …
    }
}
private static string FormatMinAndMax(string propertyName, (double Min, double Max) propertyMinAndMax)
{
   return $"{propertyName}\t"
        + $"{propertyMinAndMax.Min:F3}\t"
        + $"{_ propertyMinAndMax.Max:F3}\t"
        + $"{Environment.NewLine}";
}
private static string FormatMinAndMax(string propertyName, (int Min, int Max) propertyMinAndMax)
{
   return $"{propertyName}\t"
        + $"{propertyMinAndMax.Min}\t"
        + $"{propertyMinAndMax.Max}\t"
        + $"{Environment.NewLine}";
}
…
```

La ruta de salida del archivo de resultados se crea combinando la carpeta de salida registrada durante la inicialización y número de capa y láser del código procesado. Los valores máximo y mínimo de cada propiedad `Sample` se guardan en el archivo y, a continuación, se restablecen para procesar el código de la siguiente capa.

El plug-in devuelve un resultado “Success” con el mensaje de que será auditado por DataHUB.

NOTA: DataHUB asigna a cada instancia de plug-in ejecutado una ruta de carpeta de salida exclusiva, que se utiliza para los siguientes resultados del plug-in.

6.4.5.5 Instalación final

El plug-in completo es el siguiente:

```
using System;
using System.Collections.Generic;
using System.IO;
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Types.Tokens;
using PlugIns.API.v2.Helpers;
namespace PlugIn.Tutorials.v2.Example1
{
   public class PlugIn
   {
      public static string APIVersion() => “2”;
      public static string DisplayName() => “Tutorial Example 1”;
      public string Initialise(string initialisationData)
      {
         var initialisationValues = Jobject.Parse(initialisationData);
         _outputFolder = initialisationValues[“OutputFolder”].Value<string>();
         return InitialiseReturns.Success();
      }
      public string Process(IreadOnlyCollection<Token> tokens)
      {
         foreach (var token in tokens)
            switch (token)
            {
               case Sample sample:
                  _demandX             = MinMaxForProperty(_demandX,             sample.DemandX);
                  _demandY             = MinMaxForProperty(_demandY,             sample.DemandY);
                  _demandFocus         = MinMaxForProperty(_demandFocus,         sample.DemandFocus);
                  _demandLaserPower    = MinMaxForProperty(_demandLaserPower,    sample.DemandLaserPower);
```

```
            _laserModulate       = MinMaxForProperty(_laserModulate,       sample.LaserModulate);
            _meltVIEWPlasma      = MinMaxForProperty(_meltVIEWPlasma,      sample.MeltVIEWPlasma);
            _meltVIEWMeltpool    = MinMaxForProperty(_meltVIEWMeltpool,    sample.MeltVIEWMeltpool);
            _laserVIEW           = MinMaxForProperty(_laserVIEW,           sample.LaserVIEW);
            _laserOutputPower    = MinMaxForProperty(_laserOutputPower,    sample.LaserOutputPower);
            _laserBackReflection = MinMaxForProperty(_laserBackReflection, sample.LaserBackReflection);
          break;
        case EndOfLayerLaserData endOfLayerLaserData:
            var layer = endOfLayerLaserData.Layer.Value;
            var laser = endOfLayerLaserData.Laser.Value;
            var outputFilePath = Path.Combine(_outputFolder, "Min & Max for "
                                                           + $"layer {layer}, "
                                                           + $"laser {laser}"
                                                           + ".txt");
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandX",          _demandX));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandY",          _demandY));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandFocus",      _demandFocus));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserModulate",    _laserModulate));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandLaserPower",
                                                               _demandLaserPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWPlasma",
                                                               _meltVIEWPlasma));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWMeltpool",
                                                               _meltVIEWMeltpool));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserVIEW",
                                                               _laserVIEW
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserOutputPower",
                                                               _laserOutputPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserBackReflection",
                                                               _laserBackReflection));
            _demandX             = (double.MaxValue, double.MinValue);
            _demandY             = (double.MaxValue, double.MinValue);
            _demandFocus         = (double.MaxValue, double.MinValue);
            _laserModulate       = (int.MaxValue, int.MinValue);
            _demandLaserPower    = (int.MaxValue, int.MinValue);
            _meltVIEWPlasma      = (int.MaxValue, int.MinValue);
            _meltVIEWMeltpool    = (int.MaxValue, int.MinValue);
            _laserVIEW           = (int.MaxValue, int.MinValue);
            _laserOutputPower    = (int.MaxValue, int.MinValue);
            _laserBackReflection = (int.MaxValue, int.MinValue);
```

```
                return ProcessReturns.Success($"Processed layer {layer}, laser {laser}");
            }
        return ProcessReturns.SuccessWithoutInformation();
    }
    public string Shutdown() => ShutdownReturns.Success();
    private string _outputFolder;
    private (double Min, double Max) _demandX             = (double.MaxValue, double.MinValue);
    private (double Min, double Max) _demandY             = (double.MaxValue, double.MinValue);
    private (double Min, double Max) _demandFocus         = (double.MaxValue, double.MinValue);
    private (int Min, int Max)       _laserModulate       = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _demandLaserPower    = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _meltVIEWPlasma      = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _meltVIEWMeltpool    = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _laserVIEW           = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _laserOutputPower    = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _laserBackReflection = (int.MaxValue, int.MinValue);

    private static string FormatMinAndMax(string propertyName,
                                          (double Min, double Max) propertyMinAndMax)
    {
        return $"{propertyName}\t"
             + $"{propertyMinAndMax.Min:F3}\t"
             + $"{propertyMinAndMax.Max:F3}\t"
             + $"{Environment.NewLine}";
    }
    private static string FormatMinAndMax(string propertyName,
                                          (int Min, int Max) propertyMinAndMax)
    {
        return $"{propertyName}\t"
             + $"{propertyMinAndMax.Min}\t"
             + $"{propertyMinAndMax.Max}\t"
             + $"{Environment.NewLine}";
    }
    private static (double, double) MinMaxForProperty((double Min, double Max) currentValue,
                                                      double value)
    {
        var (min, max) = currentValue;
        return (Math.Min(min, value), Math.Max(max, value));
    }
    private static (int, int) MinMaxForProperty((int Min, int Max) currentValue,
```

```
                                                int value)
        {
            var (min, max) = currentValue;
            return (Math.Min(min, value), Math.Max(max, value));
        }
    }
}
```

6.4.6 Filtros

Un *filtro* de API es una unidad de código que procesa código de entrada y genera código adicional o ninguno. Después de procesarse correctamente, un filtro puede generar el código de entrada, nuevo código o, incluso, absorber el código de entrada. Mediante estos métodos, un filtro transforma una secuencia de código de entrada en una nueva secuencia de código de salida.

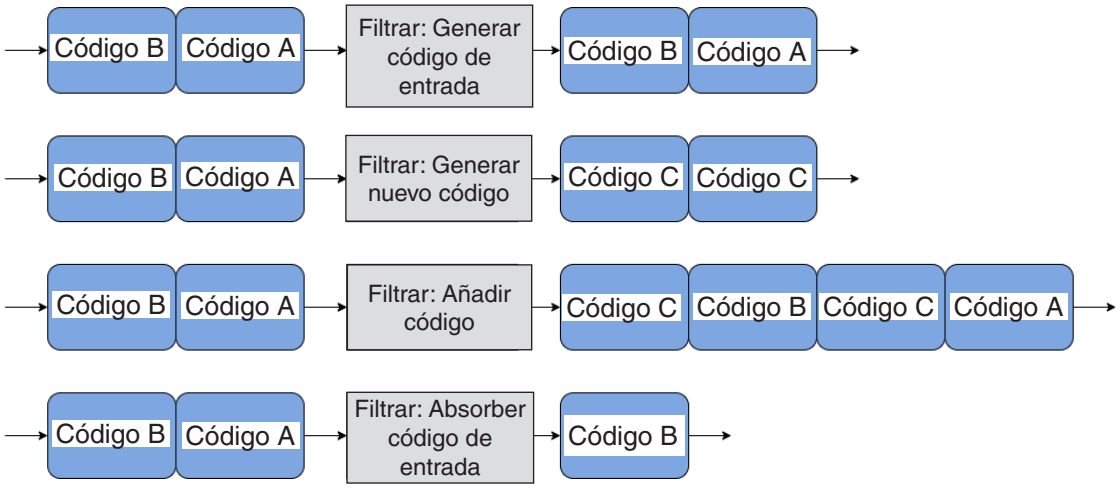


Imagen 39 Métodos de filtrado de salida

6.4.6.1 Canales

Si son dos filtros, la API proporciona un método para combinarlos en un *canal*. El hecho de que una canal tenga la misma firma que un filtro individual significa que es posible combinar los canales y filtros en canales más largos y complejos. Cuando un canal procesa una secuencia de código, la salida del primer filtro forma la entrada del segundo, y así sucesivamente, hasta que la salida del último filtro crea el resultado final del canal.

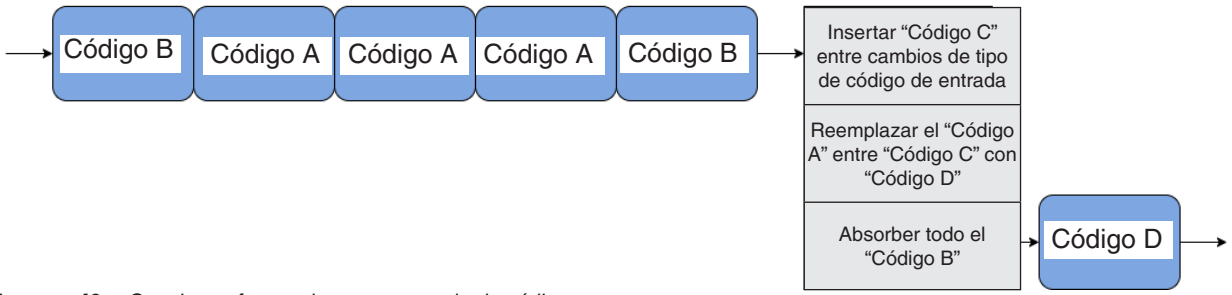


Imagen 40 Canal transformando una secuencia de código

6.4.6.2 Agrupación de muestras en paquetes mediante una canal de filtros

Esta sección describe la transformación de una secuencia de código de entrada de muestras, a través de varias etapas de filtrado, en una secuencia de código de salida de los datos agregados.

Las series de filtros que forman el canal son:

- División cuando Laser Modulate cambia
- Emisión si el láser está disparando
- División cuando DemandX cambia
- División cuando DemandY cambia
- Recopilación en paquetes de muestras
- Paquetes resumidos de muestras

6.4.6.2.1 Secuencia de código de entrada

La secuencia de código de entrada se compone de varios códigos de muestra:

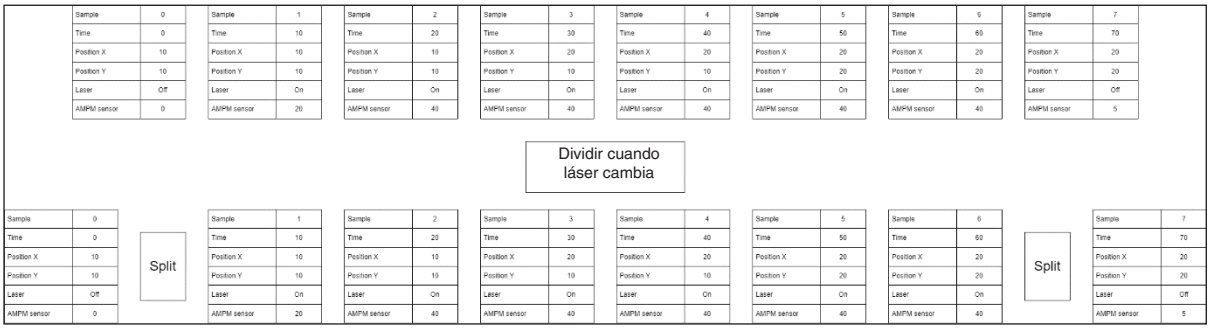
Sample	0	Sample	1	Sample	2	Sample	3	Sample	4	Sample	5	Sample	6	Sample	7
Time	0	Time	10	Time	20	Time	30	Time	40	Time	50	Time	60	Time	70
Position X	10	Position X	10	Position X	10	Position X	20	Position X	20	Position X	20	Position X	20	Position X	20
Position Y	10	Position Y	10	Position Y	10	Position Y	10	Position Y	10	Position Y	20	Position Y	20	Position Y	20
Laser	Off	Laser	On	Laser	On	Laser	On	Laser	On	Laser	On	Laser	On	Laser	Off
AMPM sensor	0	AMPM sensor	20	AMPM sensor	40	AMPM sensor	40	AMPM sensor	40	AMPM sensor	40	AMPM sensor	40	AMPM sensor	5

Puede hacer un seguimiento del funcionamiento del láser mediante distintos valores en cada muestra:

- El láser empieza en la posición (10, 10) a la hora 0, y no se dispara
- El láser dispara a la hora 20
- El láser se desplaza a la posición (20, 10) a la hora 30
- El láser se desplaza a la posición (20, 20) a la hora 50
- El láser detiene el disparo a la hora 70

6.4.6.2.2 Filtro 1: División cuando LaserModulate cambia

El primer filtro inserta código `Split` entre los cambios del canal modulado del láser:



6.4.6.2.3 Filtro 2: Emisión si el láser está disparando

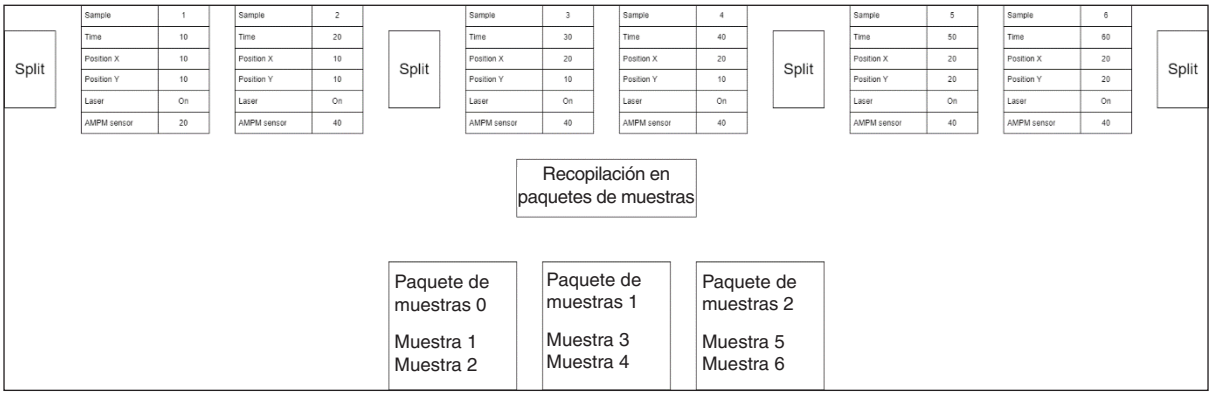
El siguiente filtro retira las muestras cuando el láser está apagado:

6.4.6.2.4 Filtros 3 y 4: División cuando la posición cambia

Los dos filtros siguientes insertan código `Split` entre las muestras cuando cambia la posición del láser:

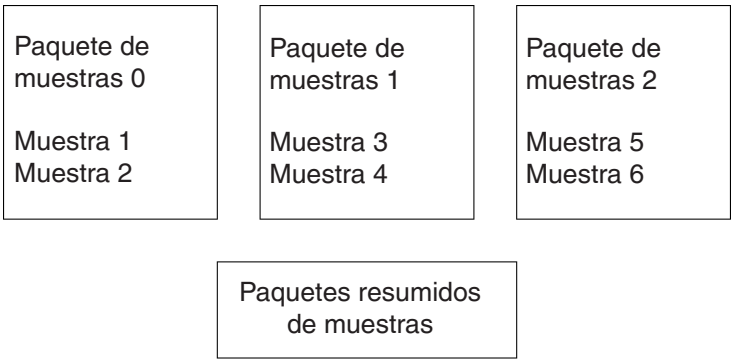
6.4.6.2.5 Filtro 5: Recopilación en paquetes de muestras

Las muestras se agrupan por código `Split`, y entre ellas se incluyen las muestras del láser disparando o apagado, por ejemplo, las muestras de un meltpool individual. Estos grupos de muestras están intercalados.



6.4.6.2.6 Filtro 6: Paquetes resumidos de muestras

Para finalizar, se resumen las muestras recopiladas:



Summary Packet	0
Time	10
Duration	20
Position X	10
Position Y	10
AMPM summary	30

Summary Packet	1
Time	30
Duration	20
Position X	20
Position Y	10
AMPM summary	30

Summary Packet	2
Time	50
Duration	20
Position X	20
Position Y	20
AMPM summary	30

Estos paquetes finales de código resumidos contienen valores que describen el tiempo de inicio (tiempo desde la primera muestra válida entregada), la duración (número de muestras válidas × duración de una muestra individual), la posición y el resumen de los valores de control del proceso.

6.4.7 Ejemplo 2: Filtros

Este ejemplo amplía el uso mínimo/máximo plug-in mediante los filtros de API. Abarcar los siguientes conceptos principales:

- Filtros
- Procesamiento de código a través de un filtro

6.4.7.1 Filtros

Los filtros procesan y modifican una secuencia de código, donde añaden, eliminan, reenvían o modifican el código. Un filtro procesa código individual y genera código adicional o ninguno. Esta asignación de uno a varios proporciona un exhaustivo mecanismo de modificación de una secuencia de código durante el proceso.

En muchas tareas de análisis de control de procesos, solo se procesan determinadas muestras grabadas cuando el láser está disparando (ya que contienen datos sobre el baño de fusión Meltpool). El filtrado de muestras grabadas cuando el láser no está disparando reduce también la cantidad de código utilizado para el análisis. Para esta finalidad, se usa el filtro integrado `EmitSampleIf.LaserModulate.IsOn`. A través de este filtro, el plug-in puede enviar el código de `Muestra` para eliminar las muestras con el láser apagado y, por tanto, producir los valores mínimo y máximo relevantes para la formación de Meltpool.

En C#, un delegado es un tipo que representa un método con una firma concreta. Un valor delegado se asigna mediante una *instancia de método*, que puede invocarse a través de un delegado.

El nuevo `namespace PlugIn.Tutorials.v2.Example2` y la clase de plug-in se añaden con el delegado`FilterOutLaserOffSamples` asignado al filtro `EmitSampleIf.LaserModulate.IsOn()` al crear la instancia de plug-in:

```
using System;
using System.Collections.Generic;
using System.IO;
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Filters;
using PlugIns.API.v2.Helpers;
using PlugIns.API.v2.Types.Tokens;
namespace PlugIn.Tutorials.v2.Example2
{
   public class PlugIn
   {
      public static string APIVersion() => “2”;
      public static string DisplayName() => “Tutorial Example 2”;
      public string Initialise(string initialisationData)
      {
         var initialisationValues = JObject.Parse(initialisationData);
         _outputFolder = initialisationValues[“OutputFolder”].Value<string>();
         return InitialiseReturns.Success();
      }
```

```
    public string Process(IReadOnlyCollection<Token> tokens) => ProcessReturns.Success();
    public string Shutdown() => ShutdownReturns.Success();
    private string _outputFolder;
    private Func<Token, IEnumerable<Token>> FilterOutLaserOffSamples { get; }
        = EmitSampleIf.LaserModulate.IsOn();
  }
}
```

Seguidamente, se instala el método Process, con una expresión LINQ para transferir cada lote de código, primero a través del filtro (mediante el delegado) y, a continuación, a través de un método que procesa la secuencia de código modificada resultante:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
  var messages = tokens.FilteredBy(FilterOutLaserOffSamples).
                        CollateMessagesFrom(ProcessTokens);
  return ProcessReturns.Success(messages);
}
private IEnumerable<string > ProcessTokens(IEnumerable<Token> tokens)
{
  foreach (var token in tokens)
    switch (token)
    {
      case Sample sample:
        _demandX             = MinMaxForProperty(_demandX,             sample.DemandX);
        _demandY             = MinMaxForProperty(_demandY,             sample.DemandY);
        _demandFocus         = MinMaxForProperty(_demandFocus,         sample.DemandFocus);
        _laserModulate       = MinMaxForProperty(_laserModulate,       sample.LaserModulate);
        _demandLaserPower    = MinMaxForProperty(_demandLaserPower,    sample.DemandLaserPower);
        _meltVIEWPlasma      = MinMaxForProperty(_meltVIEWPlasma,      sample.MeltVIEWPlasma);
        _meltVIEWMeltpool    = MinMaxForProperty(_meltVIEWMeltpool,    sample.MeltVIEWMeltpool);
        _laserVIEW           = MinMaxForProperty(_laserVIEW,           sample.LaserVIEW);
        _laserOutputPower    = MinMaxForProperty(_laserOutputPower,    sample.LaserOutputPower);
        _laserBackReflection = MinMaxForProperty(_laserBackReflection, sample.LaserBackReflection);
        break;
      case EndOfLayerLaserData endOfLayerLaserData:
        var layer = endOfLayerLaserData.Layer.Value;
        var laser = endOfLayerLaserData.Laser.Value;
        var outputFilePath = Path.Combine(_outputFolder, “Min & Max for “
                                                  + $”layer {layer}, “
```

```
                                                + $"laser {laser}"
                                                + ".txt");
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandX",             _demandX));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandY",             _demandY));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandFocus",         _demandFocus));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserModulate",       _laserModulate));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandLaserPower",    _demandLaserPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWPlasma",      _meltVIEWPlasma));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWMeltpool",    _meltVIEWMeltpool));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserVIEW",           _laserVIEW));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserOutputPower",    _laserOutputPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserBackReflection", _laserBackReflection));
            _demandX             = (double.MaxValue, double.MinValue);
            _demandY             = (double.MaxValue, double.MinValue);
            _demandFocus         = (double.MaxValue, double.MinValue);
            _laserModulate       = (int.MaxValue, int.MinValue);
            _demandLaserPower    = (int.MaxValue, int.MinValue);
            _meltVIEWPlasma      = (int.MaxValue, int.MinValue);
            _meltVIEWMeltpool    = (int.MaxValue, int.MinValue);
            _laserVIEW           = (int.MaxValue, int.MinValue);
            _laserOutputPower    = (int.MaxValue, int.MinValue);
            _laserBackReflection  = (int.MaxValue, int.MinValue);
            yield return "Processed layer {layer}, laser {laser}";
        }
}
```

NOTA: Los distintos campos (`_demandX`, etc.) y los métodos de ayuda (`FormatMinAndMax`, etc.) se omiten por brevedad y se instalan como en el primer ejemplo.

La expresión LINQ incluye dos fases. En primer lugar, se transfiere cada te de código al filtro `FilterOutLaserOffTokens` a través la API de ayuda `FilteredBy`. A continuación, la secuencia de código resultante (que solo registra código individual `Sample` de muestras grabadas cuando el láser se está disparando) se transfiere a `ProcessToken`, que realiza el procesamiento específico de cada tipo de código de interés, por ejemplo, `Sample` y `EndOfLayerLaserData`. Las cadenas resultantes devueltas por `ProcessTokens` se recopilan y devuelven a DataHUB mediante el método de ayuda `Success`.

NOTA: La sintaxis LINQ de C#' se ha diseñado para procesar secuencias de datos.

NOTA: Las secuencias de código se pueden modificar a través de un filtro.

NOTA: Un filtro devuelve código adicional o ninguno por cada entrada.

El plug-in rellena los archivos de resultados con los valores de canal mínimo y máximo de las muestras grabadas cuando el láser se está disparando.

6.4.7.2 Instalación final

El plug-in completo es el siguiente:

```
using System;
using System.Collections.Generic;
using System.IO;
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Filters;
using PlugIns.API.v2.Helpers;
using PlugIns.API.v2.Types.Tokens;
namespace PlugIn.Tutorials.v2.Example2
{
   public class PlugIn
   {
      public static string APIVersion() => “2”;
      public static string DisplayName() => “Tutorial Example 2”;
      public string Initialise(string initialisationData)
      {
         var initialisationValues = JObject.Parse(initialisationData);
         _outputFolder = initialisationValues[“OutputFolder”].Value<string>();
         return InitialiseReturns.Success();
      }
      public string Process(IReadOnlyCollection<Token> tokens)
      {
         var messages = tokens.FilteredBy(FilterOutLaserOffSamples).
                              CollateMessagesFrom(ProcessTokens);
         return ProcessReturns.Success(messages);
      }
      private IEnumerable<string> ProcessTokens(IEnumerable<Token> tokens)
      {
        foreach (var token in tokens)
         switch (token)
         {
            case Sample sample:
               _demandX             = MinMaxForProperty(_demandX,             sample.DemandX);
               _demandY             = MinMaxForProperty(_demandY,             sample.DemandY);
               _demandFocus         = MinMaxForProperty(_demandFocus,         sample.DemandFocus);
               _laserModulate       = MinMaxForProperty(_laserModulate,       sample.LaserModulate);
```

```
            _demandLaserPower    = MinMaxForProperty(_demandLaserPower,    sample.DemandLaserPower);
            _meltVIEWPlasma      = MinMaxForProperty(_meltVIEWPlasma,      sample.MeltVIEWPlasma);
            _meltVIEWMeltpool    = MinMaxForProperty(_meltVIEWMeltpool,    sample.MeltVIEWMeltpool);
            _laserVIEW           = MinMaxForProperty(_laserVIEW,           sample.LaserVIEW);
            _laserOutputPower    = MinMaxForProperty(_laserOutputPower,    sample.LaserOutputPower);
            _laserBackReflection = MinMaxForProperty(_laserBackReflection, sample.LaserBackReflection);
            break;
        case EndOfLayerLaserData endOfLayerLaserData:
            var layer = endOfLayerLaserData.Layer.Value;
            var laser = endOfLayerLaserData.Laser.Value;
            var outputFilePath = Path.Combine(_outputFolder, "Min & Max for "
                                                           + $"layer {layer}, "
                                                           + $"laser {laser}"
                                                           + ".txt");
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandX",          _demandX));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandY",          _demandY));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandFocus",      _demandFocus));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserModulate",    _laserModulate));
            File.AppendAllText(outputFilePath, FormatMinAndMax("DemandLaserPower", _demandLaserPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWPlasma",   _meltVIEWPlasma));
            File.AppendAllText(outputFilePath, FormatMinAndMax("MeltVIEWMeltpool", _meltVIEWMeltpool));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserVIEW",        _laserVIEW));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserOutputPower", _laserOutputPower));
            File.AppendAllText(outputFilePath, FormatMinAndMax("LaserBackReflection",
                                                               _laserBackReflection));
            _demandX             = (double.MaxValue, double.MinValue);
            _demandY             = (double.MaxValue, double.MinValue);
            _demandFocus         = (double.MaxValue, double.MinValue);
            _laserModulate       = (int.MaxValue, int.MinValue);
            _demandLaserPower    = (int.MaxValue, int.MinValue);
            _meltVIEWPlasma      = (int.MaxValue, int.MinValue);
            _meltVIEWMeltpool    = (int.MaxValue, int.MinValue);
            _laserVIEW           = (int.MaxValue, int.MinValue);
            _laserOutputPower    = (int.MaxValue, int.MinValue);
            _laserBackReflection = (int.MaxValue, int.MinValue);
            yield return $"Processed layer {layer}, laser {laser}";
            break;
    }
}
public string Shutdown() => ShutdownReturns.Success();
```

```
    private string _outputFolder;
    private Func<Token, IEnumerable<Token>> FilterOutLaserOffSamples { get; }
       = EmitSampleIf.LaserModulate.IsOn();
    private (double Min, double Max) _demandX             = (double.MaxValue, double.MinValue);
    private (double Min, double Max) _demandY             = (double.MaxValue, double.MinValue);
    private (double Min, double Max) _demandFocus         = (double.MaxValue, double.MinValue);
    private (int Min, int Max)       _laserModulate       = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _demandLaserPower    = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _meltVIEWPlasma      = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _meltVIEWMeltpool    = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _laserVIEW           = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _laserOutputPower    = (int.MaxValue, int.MinValue);
    private (int Min, int Max)       _laserBackReflection = (int.MaxValue, int.MinValue);
    private static string FormatMinAndMax(string propertyName, (double Min, double Max) propertyMinAndMax)
    {
       return $"{propertyName}\t"
            + $"{propertyMinAndMax.Min:F3}\t"
            + $"{propertyMinAndMax.Max:F3}\t"
            + $"{Environment.NewLine}";
    }
    private static string FormatMinAndMax(string propertyName, (int Min, int Max) propertyMinAndMax)
    {
       return $"{propertyName}\t"
            + $"{propertyMinAndMax.Min}\t"
            + $"{propertyMinAndMax.Max}\t"
            + $"{Environment.NewLine}";
    }
    private static (double, double) MinMaxForProperty((double Min, double Max) currentValue, double value)
    {
       var (min, max) = currentValue;
       return (Math.Min(min, value), Math.Max(max, value));
    }
    private static (int, int) MinMaxForProperty((int Min, int Max) currentValue, int value)
    {
       var (min, max) = currentValue;
       return (Math.Min(min, value), Math.Max(max, value));
    }
  }
}
```

6.4.8 Ejemplo 3: ExportPackets

Los filtros individuales son útiles, pero su máximo rendimiento se obtiene cuando se combinan en canales (consulte “Canales” en la página 6-56). Un canal realiza una operación compleja en una secuencia de código a través de una serie de filtros sencillos reutilizables. El plug-in `ExportPackets` incluido en el servicio DataHUB realiza esta operación, cuya instalación se recrea en este ejemplo, que abarca:

- Combinación de filtros en un canal.
- Procesamiento de una secuencia de código a través de un canal.

6.4.8.1 Combinación de filtros

El plug-in API proporciona dos métodos de ayuda `First` y `Then` para facilitar la combinación de filtros en canales. Un canal se crea de la siguiente forma:

```
First(<filter>).Then(<filter>).Then(<filter>)…Then(<filter>);
```

Es posible encadenar varios filtros y el canal resultante actúa como un filtro, que asigna un código de entrada y genera código adicional o ninguno.

El plug-in *ExportPackets* incluido en DataHUB procesa una secuencia de código en paquetes, donde un paquete es un resumen de los datos de control de procesos obtenidos para muestras consecutivas mientras el láser está disparando a una posición X e Y de demanda concreta. Para crear paquetes de muestras, la secuencia de código se procesa de la siguiente forma:

1. Divide la secuencia cada vez que cambia el estado encendido/apagado del láser.
2. Elimina todas las muestras con el láser apagado.
3. Divide la secuencia cada vez que cambia la posición X.
4. Divide la secuencia cada vez que cambia la posición Y.
5. Reúne cada juego de muestras divididas-agrupadas como paquete de muestras.
6. Genera valores de resumen promedio y mediano de cada paquete de muestras.

NOTA: Las muestras con el láser apagado de la secuencia no se filtran inicialmente, ya que pueden parecer válidas en principio y, entonces, se crearían paquetes erróneos si el láser mantiene la misma demanda X e Y cuando se enciende y apaga repetidamente.

El código inicial del plug-in es:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
```

```
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Filters;
using PlugIns.API.v2.Helpers;
using PlugIns.API.v2.Types.Tokens;
using static PlugIns.API.v2.Helpers.PositionHelpers;
using static PlugIns.API.v2.Helpers.Pipeline;
namespace PlugIn.Tutorials.v2.Example3
{
   public class PlugIn
   {
      public static string APIVersion() => "2";
      public static string DisplayName() => "Tutorial 3 - Export Packets";
      public string Initialise(string initialisationData)
      {
          _outputFolder = JObject.Parse(initialisationData)["OutputFolder"].Value<string>();
          SetColumnHeadings(_builder);
          return InitialiseReturns.Success();
      }
      public string Process(IReadOnlyCollection<Token> tokens) => ProcessReturns.Success();
      public string Shutdown() => ShutdownReturns.Success();
      private static void SetColumnHeadings(StringBuilder builder)
      {
         builder.Clear();
         builder.AppendLine("Start time\tDuration\t"
                            + "Demand X\t"
                            + "Demand Y\t"
                            + "Demand focus\t"
                            + "Demand laser power (mean)\t"
                            + "MeltVIEW plasma (mean)\t"
                            + "MeltVIEW melt pool (mean)\t"
                            + "LaserVIEW (mean)\t"
                            + "Laser back reflection (mean)\t"
                            + "Laser output power (mean)\t"
                            + "Demand laser power (median)\t"
                            + "MeltVIEW plasma (median)\t"
                            + "MeltVIEW melt pool (median)LaserVIEW (median)\t"
                            + "Laser back reflection (median)\t"
                            + "Laser output power (median)\t");
      }
      private readonly StringBuilder _builder = new StringBuilder();
```

```
      private string _outputFolder;
   }
}
```

El plug-in utiliza `StringBuilder` para acumular los resúmenes de todos los paquetes de muestras, pero antes añade los títulos de las columnas del archivo csv.

Se crea el canal explicado anteriormente y se asigna al delegado `CollateIntoPackets`:

```
…
private Func<Token, IEnumerable<Token>> CollateIntoPackets { get; }
   = First(SplitWhen.LaserModulateChanges()).
     Then(EmitSampleIf.LaserModulate.IsOn()).
     Then(SplitWhen.DemandXChanges()).
     Then(SplitWhen.DemandYChanges()).
     Then(CollateInto.PacketOfSamples()).
     Then(SummarisePacketOfSamples.Summarise(new SummarisePacketOfSamples.SummaryMethod(
                                              "Mean",
                                              values => values.Average(x => x)),
                                            new SummarisePacketOfSamples.SummaryMethod(
                                              "Median",
                                              Median)));
private static double Median(IReadOnlyList<double> values)
{
   var halfwayIndex = values.Count >> 1;
   return values.Count % 2 == 1
            ? values[halfwayIndex]
            : (values[halfwayIndex] + values[halfwayIndex - 1]) * 0.5;
}
…
```

El canal se compone de varios filtros encadenados integrados, entonces, el resultado final del canal es una serie de códigos `SummaryPacket` que contienen los valores promedio y mediano. Para ver una descripción general de la función del canal, consulte “Agrupación de muestras en paquetes mediante una canal de filtros” en la página 6-57.

OBSERVE: Los filtros individuales encadenados forman un canal que puede realizar una transformación compleja de una secuencia de código de entrada.

El filtro final, `SummarisePacketOfSamples.Summarise`, se inicia con `SummaryMethods` que asigna el nombre al resumen – “Promedio” y “Mediano” – y proporciona el método por el que se calcula el valor. Es una forma flexible de configurar cómo se resumen los paquetes de muestras: puede utilizar diversos métodos de resumen, por ejemplo, generar promedios, medianos, mínimos y máximos:

```
SummarisePacketOfSamples.Summarise(
   new SummarisePacketOfSamples.SummaryMethod(“Mean”,    values => values.Average(x => x)),
   new SummarisePacketOfSamples.SummaryMethod(“Median”,  Median)),
   new SummarisePacketOfSamples.SummaryMethod(“Minimum”, values => values.Min(x => x)),
   new SummarisePacketOfSamples.SummaryMethod(“Maximum”, values => values.Max(x => x)))
```

El paso final consiste en instalar el método *Process*, que añade (mediante el método de ayuda `AddSummaryLine`) a `StringBuilder` una línea por cada código `SummaryPacket` y escribe el contenido de `StringBuilder` en el archivo de salida:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
   var messages = tokens.FilteredBy(CollateIntoPackets).
                         CollateMessagesFrom(ProcessToken);
   return ProcessReturns.Success(messages);
}
private IEnumerable<string> ProcessToken(IEnumerable<Token> tokens)
{
   foreach (var token in tokens)
      switch (token)
      {
         case SummaryPacket summaryPacket:
            AddSummaryLine(_builder,
                           summaryPacket.Time,
                           summaryPacket.Duration,
                           summaryPacket.Summary[“Mean”],
                           summaryPacket.Summary[“Median”]);
         break;
         case EndOfLayerLaserData endOfLayerLaserData:
            var layer = endOfLayerLaserData.Layer.Value;
            var laser = endOfLayerLaserData.Laser.Value;
            File.AppendAllText(Path.Combine(_outputFolder,
                               $”Packet data for layer {layer}, laser {laser}.txt”), _builder.ToString());
            SetColumnHeadings(_builder);
            yield return $”Processed layer {layer}, laser {laser}”;
            break;
```

```
        }
}
private void AddSummaryLine(StringBuilder builder,
                            uint time,
                            uint duration,
                            SummaryPacket.SummaryValues means,
                            SummaryPacket.SummaryValues medians)
{
   builder.AppendLine($"{time}\t"
                    + $"{duration}\t"
                    + $"{FormatTo3dp(RoundToMicrons(means.DemandX,     1))}\t"
                    + $"{FormatTo3dp(RoundToMicrons(means.DemandY,     1))}\t"
                    + $"{FormatTo3dp(RoundToMicrons(means.DemandFocus, 1))}\t"
                    + $"{FormatTo3dp(means.DemandLaserPower)}\t"
                    + $"{FormatTo3dp(means.MeltVIEWPlasma)}\t"
                    + $"{FormatTo3dp(means.MeltVIEWMeltpool)}\t"
                    + $"{FormatTo3dp(means.LaserVIEW)}\t"
                    + $"{FormatTo3dp(means.LaserBackReflection)}\t"
                    + $"{FormatTo3dp(means.LaserOutputPower)}\t"
                    + $"{FormatTo3dp(medians.DemandLaserPower)}\t"
                    + $"{FormatTo3dp(medians.MeltVIEWPlasma)}\t"
                    + $"{FormatTo3dp(medians.MeltVIEWMeltpool)}\t"
                    + $"{FormatTo3dp(medians.LaserVIEW)}\t"
                    + $"{FormatTo3dp(medians.LaserBackReflection)}\t"
                    + $"{FormatTo3dp(medians.LaserOutputPower)}\t");
}
private string FormatTo3dp(double meansDemandLaserPower)
{
   return meansDemandLaserPower.ToString("F3", _currentCulture);
}
…
```

Cada código `SummaryPacket` se añade como una línea de texto con el formato adecuado a cada valor, mediante `StringBuilder`. Cuando se detecta un código `EndOfLayerLaserData`, el plug-in guarda los datos del paquete recopilado en un archivo en la carpeta de salida.

NOTA: La secuencia de código de la construcción puede procesarse a través de un canal predefinido.

6.4.8.2 Instalación final

El plug-in completo es el siguiente:

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Filters;
using PlugIns.API.v2.Helpers;
using PlugIns.API.v2.Types.Tokens;
using static PlugIns.API.v2.Helpers.PositionHelpers;
using static PlugIns.API.v2.Helpers.Pipeline;
namespace PlugIn.Tutorials.v2.Example3
{
   public class PlugIn
   {
      public static string APIVersion() => “2”;
      public static string DisplayName() => “Tutorial 3 - Export Packets”;
      public string Initialise(string initialisationData)
      {
         _outputFolder = JObject.Parse(initialisationData)[“OutputFolder”].Value<string>();
         SetColumnHeadings(_builder);
         return InitialiseReturns.Success();
      }
      public string Process(IReadOnlyCollection<Token> tokens)
      {
         var messages = tokens.FilteredBy(CollateIntoPackets).
                              CollateMessagesFrom(ProcessToken);
         return ProcessReturns.Success(messages);
      }
      public string Shutdown() => ShutdownReturns.Success();
      private IEnumerable<string> ProcessToken(IEnumerable<Token> tokens)
      {
         foreach (var token in tokens)
            switch (token)
            {
               case SummaryPacket summaryPacket:
```

```
                AddSummaryLine(_builder,
                               summaryPacket.Time,
                               summaryPacket.Duration,
                               summaryPacket.Summary[“Mean”],
                               summaryPacket.Summary[“Median”]);
                break;
            case EndOfLayerLaserData endOfLayerLaserData:
                var layer = endOfLayerLaserData.Layer.Value;
                var laser = endOfLayerLaserData.Laser.Value;
                File.AppendAllText(Path.Combine(_outputFolder,
                                                $”Packet data for layer {layer}, laser {laser}.txt”),
                                   _builder.ToString());
                SetColumnHeadings(_builder);
                yield return $”Processed layer {layer}, laser {laser}”;
                break;
        }
    }
    private static void SetColumnHeadings(StringBuilder builder)
    {
        builder.Clear();
        builder.AppendLine(“Start time\tDuration\t”
                           + “Demand X\t”
                           + “Demand Y\t”
                           + “Demand focus\t”
                           + “Demand laser power (mean)\t”
                           + “MeltVIEW plasma (mean)\t”
                           + “MeltVIEW melt pool (mean)\t”
                           + “LaserVIEW (mean)\t”
                           + “Laser back reflection (mean)\t”
                           + “Laser output power (mean)\t”
                           + “Demand laser power (median)\t”
                           + “MeltVIEW plasma (median)\t”
                           + “MeltVIEW melt pool (median)LaserVIEW (median)\t”
                           + “Laser back reflection (median)\t”
                           + “Laser output power (median)\t”);
    }
    private void AddSummaryLine(
        StringBuilder builder,
        uint time,
        uint duration,
```

```
    SummaryPacket.SummaryValues means,
    SummaryPacket.SummaryValues medians)
{
    builder.AppendLine($"{time}\t"
                     + $"{duration}\t"
                     + $"{FormatTo3dp(RoundToMicrons(means.DemandX,     1))}\t"
                     + $"{FormatTo3dp(RoundToMicrons(means.DemandY,     1))}\t"
                     + $"{FormatTo3dp(RoundToMicrons(means.DemandFocus, 1))}\t"
                     + $"{FormatTo3dp(means.DemandLaserPower)}\t"
                     + $"{FormatTo3dp(means.MeltVIEWPlasma)}\t"
                     + $"{FormatTo3dp(means.MeltVIEWMeltpool)}\t"
                     + $"{FormatTo3dp(means.LaserVIEW)}\t"
                     + $"{FormatTo3dp(means.LaserBackReflection)}\t"
                     + $"{FormatTo3dp(means.LaserOutputPower)}\t"
                     + $"{FormatTo3dp(medians.DemandLaserPower)}\t"
                     + $"{FormatTo3dp(medians.MeltVIEWPlasma)}\t"
                     + $"{FormatTo3dp(medians.MeltVIEWMeltpool)}\t"
                     + $"{FormatTo3dp(medians.LaserVIEW)}\t"
                     + $"{FormatTo3dp(medians.LaserBackReflection)}\t"
                     + $"{FormatTo3dp(medians.LaserOutputPower)}\t");
}
private Func<Token, IEnumerable<Token>> CollateIntoPackets { get; }
    = First(SplitWhen.LaserModulateChanges()).
      Then(EmitSampleIf.LaserModulate.IsOn()).
      Then(SplitWhen.DemandXChanges()).
      Then(SplitWhen.DemandYChanges()).
      Then(CollateInto.PacketOfSamples()).
      Then(SummarisePacketOfSamples.Summarise(new SummarisePacketOfSamples.SummaryMethod(
                                                "Mean",
                                                values => values.Average(x => x)),
                                              new SummarisePacketOfSamples.SummaryMethod(
                                                "Median",
                                                Median)));
private static double Median(IReadOnlyList<double> values)
{
    var halfwayValue = values.Count >> 1;
    return values.Count % 2 == 1
        ? values[halfwayValue]
        : (values[halfwayValue] + values[halfwayValue - 1]) * 0.5;
}
```

```
      private string FormatTo3dp(double meansDemandLaserPower)
      {
         return meansDemandLaserPower.ToString(“F3”, _currentCulture);
      }
      private readonly IFormatProvider _currentCulture = CultureInfo.CurrentCulture;
      private readonly StringBuilder _builder = new StringBuilder();
      private string _outputFolder;
   }
}
```

6.4.9 Ejemplo 4: Devolución de series cronológicas de datos a Renishaw Central

El siguiente ejemplo muestra cómo el plug-in crea y completa una serie cronológica de puntos de datos en Renishaw Central. La IU de la web de Renishaw Central muestra la serie cronológica de datos en un gráfico.

El plug-in de ejemplo define una serie cronológica y, por cada capa, añade un punto de datos al valor máximo del canal LaserVIEW. La definición de la serie cronológica y sus puntos de datos se devuelven a DataHUB a través de los valores de retorno de los métodos *Initialise* y Process.

Se tratan los siguientes conceptos principales:

- Se crea una serie cronológica en Renishaw Central
- Se añaden los datos a la serie cronológica en Renishaw Central

Se inicia con el plug-in estándar vacío:

```
using System;
using System.Collections.Generic;
using System.IO;
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Types.Tokens;
using PlugIns.API.v2.Types;
namespace PlugIn.Tutorials.v2.Example4
{
   public class PlugIn
   {
      public static string APIVersion() => “2”;
      public static string DisplayName() => “Tutorial Example 4”;
      public string Initialise(string initialisationData) => InitialiseReturns.Success();
      public string Process(IReadOnlyCollection<Token> tokens) => ProcessReturns.Success();
      public string Shutdown() => ShutdownReturns.Success();
   }
}
```

6.4.9.1 Inicialización

DataHUB inicia este plug-in para devolver una definición a la serie cronológica “LaserVIEW Max”. Para definir una serie cronológica, se necesita un nombre, un tipo de unidad y la unidad. También se pueden definir propiedades adicionales, como la propiedad Grouping, que organiza las series cronológicas en un grupo utilizado por la página web de Renishaw Central para recopilar y gestionar la presentación de las series cronológicas relacionadas.

La definición de las series cronológicas se devuelve a DataHUB mediante el método `InitialiseReturns SuccessAndCreateTimeSeries()`. DataHUB convierte el valor devuelto en correos para Renishaw Central con los datos necesarios para crear las series cronológicas listas para rellenarse con puntos de datos:

```
…
public string Initialise(string initialisationData)
{
   _laserViewMax = TimeSeries.Factory().
                              Name(“LaserVIEW Max”)UnitType(“Intensity”).
                              DimensionlessUnit().
                              Grouping(“LaserVIEW”).
                              Create();
   return InitialiseReturns.SuccessAndCreateTimeSeries(_laserViewMax);
}
private TimeSeries _laserViewMax;
…
```

OBSERVE: Puede devolver una o varias definiciones de series cronológicas mediante el método Initialise del plug-in.

OBSERVE: DataHUB gestiona los detalles de los correos a Renishaw Central, de forma que los plug-in solo tienen que concentrarse en las definiciones de las series cronológicas.

6.4.9.2 Procesamiento

Al procesar el código `Sample` en la secuencia, se compara el valor máximo actual con el valor del canal LaserVIEW de la muestra y se actualiza si es necesario:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
   foreach (var token in tokens)
   {
      switch (token)
      {
         case Sample sample:
            if (sample.LaserVIEW >= _max.value)
               _max = (sample.Time, sample.LaserVIEW);
         break;
      }
   }
   return ProcessReturns.SuccessWithoutInformation();
}
private (uint time, int value) _max = (0, int.MinValue);
…
```

Al final de los datos del láser de una capa concreta, cuando el plug-in recibe el código `EndOfLayerLaserData`, devuelve un mensaje de aprobación a DataHUB para indicar que los datos de la capa y el láser se han procesado:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
   foreach (var token in tokens)
      switch (token)
      {
         …
         case EndOfLayerLaserData endOfLayerLaserData:
            return ProcessReturns.Success($”Processed layer {endOfLayerLaserData.Layer.Value}, “ +
                                          $”laser {endOfLayerLaserData.Laser.Value}”);
         …
      }
   return ProcessReturns.SuccessWithoutInformation();
}
…
```

Al final de los datos de la capa, el plug-in recibe un código `EndOfLayerData`. El objeto `TimeSeries` se utiliza para producir un objeto `UpdateTimeSeries` con un valor de series cronológicas que contiene el valor máximo final de la capa. El método de ayuda `ProcessReturns` proporciona un método `SuccessAndSendData()` que obtiene una matriz de objetos `UpdateTimeSeries`.

El valor máximo registrado de la capa se restablece para procesar el siguiente lote de código y la actualización de la serie cronológica devuelta:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
   …
   case EndOfLayerData endOfLayerData:
      var laserViewMaxUpdate = _laserViewMax.Update().
                                             WithValues((_max.Time, _max.Value)).
                                             NoLimits();
      _max = (0, int.MinValue);
      return ProcessReturns.SuccessAndSendData($"Processed layer {endOfLayerData.Layer.Value}",
                                               new[] {laserViewMaxUpdate});
   …
}
```

6.4.9.3 Instalación final

El plug-in completo es el siguiente:

```
using System.Collections.Generic;
using PlugIns.API.v2.Helpers;
using PlugIns.API.v2.Types;
using PlugIns.API.v2.Types.Tokens;
namespace PlugIn.Tutorials.v2.Example4
{
   public class PlugIn
   {
      public static string APIVersion() => "2";
      public static string DisplayName() => "Tutorial Example 4";
      public string Initialise(string initialisationData)
      {
         _laserViewMax = TimeSeries.Factory().
                                    Name("LaserVIEW Max").
                                    UnitType("Intensity").
                                    DimensionlessUnit().
                                    Grouping("LaserVIEW").
```

```
                                    Create();
        return InitialiseReturns.SuccessAndCreateTimeSeries(_laserViewMax);
    }
    public string Process(IReadOnlyCollection<Token> tokens)
    {
        foreach (var token in tokens)
            switch (token)
            {
                case Sample sample:
                    if (sample.LaserVIEW >= _max.Value)
                        _max = (sample.Time, sample.LaserVIEW);
                    break;
                case EndOfLayerLaserData endOfLayerLaserData:
                    return ProcessReturns.Success($"Processed layer {endOfLayerLaserData.Layer.Value}, "
                        + $"laser {endOfLayerLaserData.Laser.Value}");
                case EndOfLayerData endOfLayerData:
                    var laserViewMaxUpdate = _laserViewMax.Update().
                                                           WithValues((_max.Time, _max.Value)).
                                                           NoLimits();
                    _max = (0, int.MinValue);
                    return ProcessReturns.SuccessAndSendData($"Processed layer {endOfLayerData.Layer.Value}",
                                                             new[] {laserViewMaxUpdate});
            }
        return ProcessReturns.SuccessWithoutInformation();
    }
    public string Shutdown() => ShutdownReturns.Success();
    private (uint Time, int Value) _max = (0, int.MinValue);
    private TimeSeries             _laserViewMax;
  }
}
```

6.4.10 Ejemplo 5: Activación de alertas en Renishaw Central

Este ejemplo muestra cómo activar alertas en Renishaw Central cuando determinados valores superan los umbrales predefinidos.

Se tratan los siguientes conceptos principales:

- Descodificación de datos de inicialización
- Activación de alertas

6.4.10.1 Configuración del trabajo de plug-in con datos de inicialización

Los umbrales que superan los valores de alertas de este ejemplo se controlan mediante los datos de inicialización del plug-in. En este ejemplo, los datos de inicialización que coinciden con la siguiente estructura se introducen en la fase “Especificar datos de inicialización del plug-in” de DataHUB Generator.

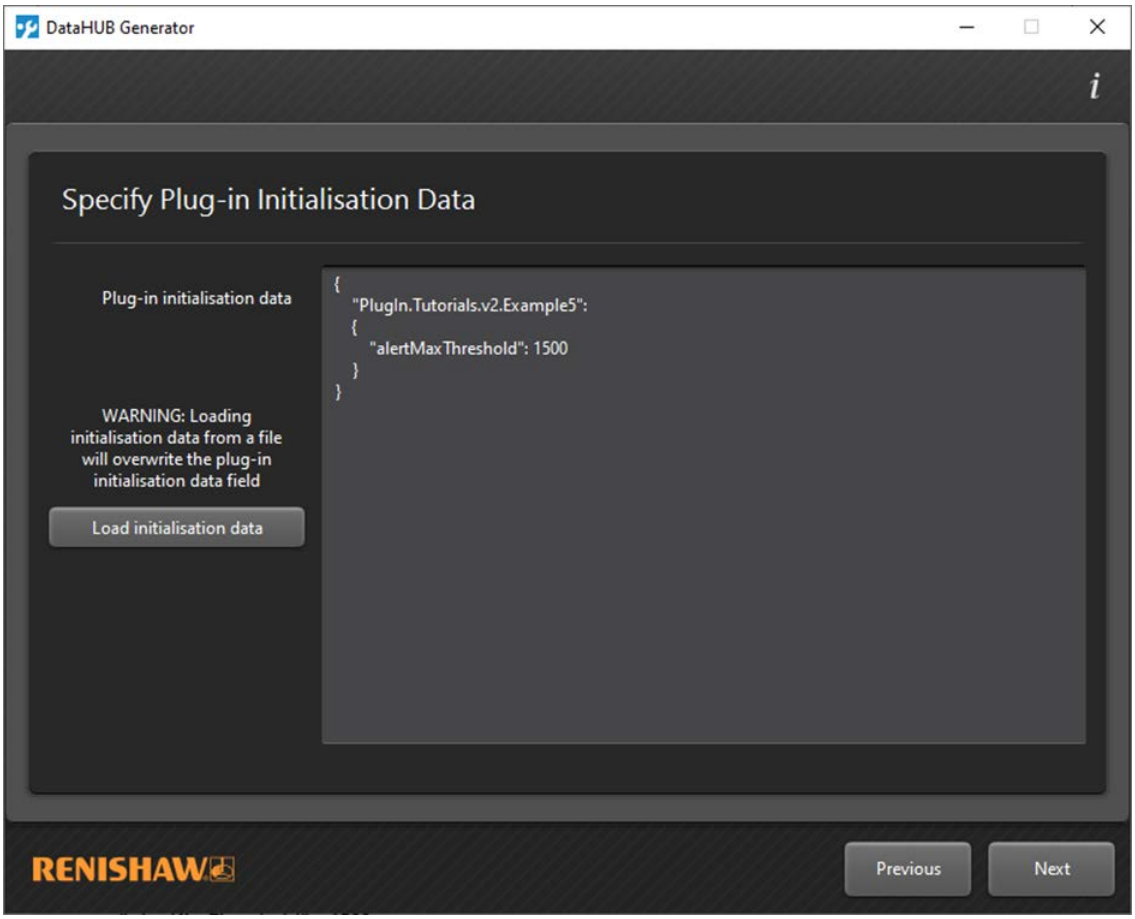


Imagen 41 Especificar datos de inicialización del plug-in en DataHUB Generator

```
{
  “PlugIn.Tutorials.v2.Example5”:
  {
    “alertMaxThreshold”: 1500
  }
}
```

Se inicia con el plug-in estándar vacío:

```
using System;
using System.Collections.Generic;
using System.IO;
using Newtonsoft.Json.Linq;
using PlugIns.API.v2.Types.Tokens;
namespace PlugIn.Tutorials.v2.Example5
{
   public class PlugIn
   {
      public static string APIVersion() => “2”;
      public static string DisplayName() => “Tutorial Example 5”;
      public string Initialise(string initialisationData) => InitialiseReturns.Success();
      public string Process(IReadOnlyCollection<Token> tokens) => ProcessReturns.Success();
      public string Shutdown() => ShutdownReturns.Success();
   }
}
```

6.4.10.2 Inicialización

Durante la inicialización, es posible acceder a los datos específicos del plug-in desde el objeto JSON “initialisationData” con el nombre proporcionado en la configuración del trabajo. En este ejemplo, “PlugIn. Tutorials.v2.Example5”.

```
…
public string Initialise(string initialisationData)
{
   var iv               = JObject.Parse(initialisationData);
   var id               = JObject.Parse(iv[“InitialisationData”].Value<string>());
   var example5InitData = id[“PlugIn.Tutorials.v2.Example5”];
   _alertMaxThreshold = example5InitData[“alertMaxThreshold”].Value<int>();
   InitialiseReturns.Success();
}
private int _alertMaxThreshold;
…
```

NOTA: Durante la inicialización, el plug-in utiliza la cadena de inicialización especificada en la configuración de la tarea de procesamiento en DataHUB Generator.

6.4.10.3 Procesamiento

Este plug-in activa una alerta en Renishaw Central en caso de que el valor máximo de LaserVIEW supere el umbral definido. En primer lugar, se registra e inicializa el valor máximo actual de la capa y la hora de la muestra asociada. Tras procesar el código de la muestra, se compara el canal “LaserVIEW” con el valor máximo actual y, si es necesario, se sobrescribe. La hora de la muestra se almacena también con el valor máximo:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
   foreach (var token in tokens)
      switch (token)
      {
         case Sample sample:
            if (sample.LaserVIEW >= _max.value)
               _max = (sample.Time, sample.LaserVIEW);
            break;
      }
   return SuccessWithoutInformation();
}
private (uint time, int value) _max = (0, int.MinValue);
…
```

Al final de los datos del láser de una capa determinada, el plug-in recibe un código `EndOfLayerLaserData`. El mensaje correcto se devuelve a DataHUB para indicar que los datos de la capa y el láser se han procesado correctamente:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
   foreach (var token in tokens)
      switch (token)
      {
         …
         case EndOfLayerLaserData endOfLayerLaserData:
            return Success($”Processed layer {endOfLayerLaserData.Layer}, “ +
                           $”laser {endOfLayerLaserData.Laser}”);
         …
      }
   return SuccessWithoutInformation();
}
…
```

Al final de los datos de la capa, el plug-in recibe un código EndOfLayerData que indica que no quedan datos pendientes de la capa. Cuando se detecta este código, se puede finalizar el proceso y enviar los resultados a Renishaw Central.

Si el máximo actual supera el umbral definido, se crea una alerta y se añade a la lista. La alerta se crea con una descripción, el nivel de importancia, el nombre, y la fecha y hora.

El método de ayuda `ProcessReturns` proporciona un método `SuccessAndSendData`, que obtiene un mensaje y una matriz de alertas y codifica los datos que se devuelven a DataHUB:

```
…
public string Process(IReadOnlyCollection<Token> tokens)
{
   foreach (var token in tokens)
      switch (token)
      {
         …
         case EndOfLayerData endOfLayerData:
            var alerts = new List<Alert>();
            if (_max.value > _alertMaxThreshold)
               alerts.Add(Alert.Factory().
                                 Descripción ("La intensidad de LaserVIEW ha superado el umbral máximo”).
                                 Warning().
                                 Name(“LaserVIEW Max Threshold”).
                                 Time(_max.time));

            _max = (0, int.MinValue);
            return ProcessReturns.
                        SuccessAndSendData($”Processed layer {endOfLayerData.Layer.Value}”,
                                          alerts);
        …
      }
…
```

NOTA: Las descripciones de alertas devueltas a DataHUB se envían a Renishaw Central, que puede activar acciones, como enviar un SMS.

6.4.10.4 Instalación final

El plug-in completo es el siguiente:

```
public class PlugIn
{
   public static string APIVersion() => “2”;
   public static string DisplayName() => “Tutorial Example 5”;
   public string Initialise(initialisationData)
   {
      var iv               = JObject.Parse(initialisationData);
      var id               = JObject.Parse(iv[“InitialisationData”].Value<string>());
      var example5InitData = id[“PlugIn.Tutorials.v2.Example5”].ToObject<InitialisationData>();
      _alertMaxThreshold = example5InitData.AlertMaxThreshold;
      return InitialiseReturns.Success();
   }
   public string Process(IReadOnlyCollection<Token> tokens)
   {
      foreach (var token in tokens)
         switch (token)
         {
            case Sample sample:
               if (sample.LaserVIEW >= _max.value)
                  _max = (sample.Time, sample.LaserVIEW);
               break;
            case EndOfLayerLaserData endOfLayerLaserData:
               return ProcessReturns.Success($”Processed layer {endOfLayerLaserData.Layer.Value}, “
                                             + $”laser {endOfLayerLaserData.Laser.Value}”);
            case EndOfLayerData endOfLayerData:
               var alerts = new List<Alert>();
               if (_max.value > _alertMaxThreshold)
                  alerts.Add(Alert.Factory().
                                    Descripción ("La intensidad de LaserVIEW ha superado el umbral máximo”).
                                    Warning().
                                    Name(“LaserVIEW Max Threshold”).
                                    Time(_max.time));
               _max = (0, int.MinValue);
               return ProcessReturns.
                       SuccessAndSendData($”Processed layer {endOfLayerData.Layer.Value}”, alerts);
         }
      return ProcessReturns.SuccessWithoutInformation();
```

```
    }
    public string Shutdown() => ShutdownReturns.Success();
    private (uint time, int value) _max = (0, int.MinValue);
    private int _alertMaxThreshold;
}
```

6.4.11 Plug-in API V2.0

Para que un conjunto sea compatible con el servicio DataHUB como plug-in V2.0, debe instalar la siguiente API pública.

6.4.11.1 Asignación de nombres del conjunto

El nombre de archivo del conjunto .NET debe empezar por “PlugIn.” (“PlugIn” seguido de un punto) y tener la extensión “dll”.

6.4.11.2 Asignación de nombres de la clase de plug-in

El conjunto de plug-in debe definir una clase pública denominada “PlugIn”.

6.4.11.3 Métodos de la clase de plug-in

La clase de plug-in debe instalar los siguientes métodos públicos:

```
public PlugIn()
public static string APIVersion()
public static string DisplayName()
public string Initialise(string data)
public string Process(IReadOnlyCollection<Token> data)
public string Shutdown()
```

6.4.11.4 Constructor sin parámetros

El tipo debe incluir un constructor sin parámetros. Si no se han definido constructores sin parámetros, C# los genera automáticamente, por tanto, no es necesario definirlos expresamente.

NOTA: Un plug-in no necesita ningún recurso en este constructor.

Una instancia de plug-in construida no garantiza la llamada a *Shutdown*, por tanto, es posible que estos recursos no se eliminen a tiempo. *Initialise* es la ubicación adecuada para adquirir recursos.

6.4.11.5 Método `APIVersion` estático

El método `APIVersion` estático de DataHUB identifica la versión de API en la que se debe validar el plug-in y el formato de datos de control de procesos que debe gestionar.

Parámetros:

Ninguno.

Devuelve: `string`

Un plug-in que instala la API V2.0 debe devolver el literal “2”.

6.4.11.6 Método `DisplayName` estático

El contenido de la cadena devuelta por el método `DisplayName` se utiliza al mostrar el nombre mostrado del plug-in en DataHUB Monitor, al crear la carpeta de salida del plug-in, auditar los eventos, etc. No es obligatorio que este nombre sea exclusivo, pero si lo es, facilita la identificación de plug-in, por ejemplo, cuando hay varias versiones instaladas en un DCPC.

Parámetros:

Ninguno.

Devuelve: `string`

Literal para usar en los nombres de plug-in en distintos DataHUB UX y registros.

6.4.11.7 Método `Initialise`

DataHUB invoca este método al iniciar una construcción, antes de enviar datos de capas al plug-in. Por tanto, pasa los datos de la *construcción sin variar* al plug-in. El plug-in puede utilizar este método para asignar los recursos necesarios durante la vida útil de la instancia, calcular constantes de la construcción, etc.

Parámetro: `string`

Cadena que contiene un objeto JSON conforme al esquema siguiente:

```
{
   “AssemblyFolder”:     string
   “OutputFolder”:       string
   “NumberOfLasers”:     unsigned integer
   “InitialisationData”: JSON object
}
```

`AssemblyFolder: string`

Ruta de la carpeta que contiene el conjunto de este plug-in:

“C:\ProgramData\Renishaw\DataHUB\PlugIns\PlugIn.Example”

`OutputFolder: string`

Ruta de la carpeta en la que la instancia del plug-in guarda los archivos de salida. Cada instancia del plug-in ejecutada para un trabajo utiliza una carpeta exclusiva generada en la carpeta de salida de los trabajos. El nombre de la carpeta se compone del nombre del plug-in, la versión de API, la fecha y hora, y la GUID, con la siguiente estructura:

“\[2] <Nombre del plug-in>.<Fecha y hora con formato yyyMMdd-hhmmss.fff> (<GUID>)”

`NumberOfLasers: unsigned integer`

Es un valor del 1 a 4, como se define en la configuración de hardware de la máquina que genera los datos.

InitialisationData: JSON object

Contenido facilitado en la fase “Datos de inicialización del plug-in” del proceso de creación del trabajo en DataHUB Generator. Para obtener más información sobre la estructura y el uso de esta cadena, consulte “Cadena de datos de inicialización” en la página 6-97.

Devuelve: string

Cadena que contiene un objeto JSON conforme al esquema siguiente:

```
{
   “result”: string
   “message”: string
   “timeSeries”:
   [
      {
         “name”:               string
         “unitType”:           string
         “unit”:               string
         “absoluteOrRelative”: string
         “component”:          string
         “displayPrecision”:   unsigned integer
         “graphType”:          string
         “grouping”:           string
         “precision”:          unsigned integer
         [Cualquier número de propiedades adicionales personalizadas]
       }
    ]
 }
```

result: string

Indica si el plug-in se ha inicializado correctamente o no. Debe ser:

- el valor literal “Success”, que indica que el plug-in se ha inicializado correctamente y puede iniciar el proceso, o bien
- el valor literal “Failure”, que indica que el plug-in ha experimentado un error y se apagará sin recibir datos.

Esta propiedad es obligatoria.

message: string

Mensaje con más detalles que se adjunta al resultado y se guarda en el registro de auditoría. Esta propiedad es opcional, si no se incluye, solo se guarda el resultado en el registro de auditoría.

timeSeries: JSON array

Matriz que contiene las definiciones de la serie cronológica que usa el plug-in para publicar los datos durante el procesamiento. Esta propiedad es opcional, si no se incluye o la matriz está vacía, DataHUB no crea ninguna serie cronológica en Renishaw Central.

timeSeries[n].name: string

Nombre de la serie cronológica. Esta propiedad es obligatoria.

timeSeries[n].unitType: string

Familia de unidades que representan los datos. Permite convertir el proceso al tipo de unidades que desea utilizar. Los tipos de unidades por defecto son:

- Acceleration
- Angle
- Binary
- Distance
- Flowrate
- Humidity
- MicroDistance
- Percentage
- Pressure
- Speed
- Temperature
- Dimensionless

Esta propiedad es obligatoria.

timeSeries[n].unit: string

Generalmente, Renishaw Central prefiere unidades SI, salvo que se indiquen relaciones “por” mediante el carácter "/" y el formato de potencias con número. Por ejemplo, la aceleración se representa normalmente como m/s^2. Para temperatura y grados, utilice el carácter “°” donde proceda, por ejemplo, °C para Celsius. Si los valores no tienen dimensión, utilice el valor literal `Dimensionless`. Esta propiedad es obligatoria.

timeSeries[n].absoluteOrRelative: string

Sugerencia sobre si los valores de la aplicación deben absolutos o relativos a un valor anterior. Debe ser:

- el valor literal “Absolute”, que indica que los valores son absolutos, o bien
- el valor literal “Relative”, que indica que los valores son relativos al valor anterior.

Esta propiedad es opcional, si no se especifica, se aplica el valor Absoluto.

timeSeries[n].component: string

Asocia la serie cronológica a un aspecto distintivo de un dispositivo, generalmente, una entidad física, como una estación meteorológica, una herramienta o un sistema. Esta propiedad es opcional, si no se especifica, se aplica el valor "Máquina".

`timeSeries[n].description: string`

Descripción de la serie cronológica. Esta propiedad es opcional.

`timeSeries[n].displayHintPrecision: unsigned integer`

Sugerencia sobre los valores de precisión de la serie cronológica que muestran las aplicaciones que integran Renishaw Central. Esta propiedad es opcional.

`timeSeries[n].displayHintSampleRate: string`

Sugerencia sobre la frecuencia de muestreo que presentan las aplicaciones que integran Renishaw Central. Si no se especifica, la mayoría de las aplicaciones asumen el uso "por defecto". Las frecuencias de muestreo más conocidas son:

- Raw
- Default
- Hour
- Day
- Week
- Month

Esta propiedad es opcional.

`timeSeries[n].graphType: string`

Sugerencia sobre el tipo de gráfico de la serie cronológica que utilizan las aplicaciones que integran Renishaw Central. Los tipos de gráficos más conocidos son:

- Bar
- Binary (específico para datos Activados-Desactivados)
- Line

Esta propiedad es opcional.

`timeSeries[n].grouping: string`

Grupo en el que se obtienen esta serie cronológica. Esta propiedad es opcional.

`timeSeries[n].precision: unsigned integer`

Registro de la precisión de los datos capturados y calculados. Esta propiedad es opcional.

`timeSeries[n]` *Propiedades adicionales*

El plug-in puede incluir cualquier cantidad de propiedades adicionales para almacenar en Renishaw Central, a las que se puede acceder con software personalizado. No puede utilizar los nombres reservados “machine” o “source”.

Si la cadena devuelta no cumple el esquema anterior, se trata como un error y se guarda la cadena completa en el registro de auditoría.

6.4.11.8 Método Process

DataHUB transfiere las subsecciones de los datos de construcción en llamadas secuenciales: el número total de llamadas varía según el tamaño de los datos. En este método, el plug-in debe realizar el volumen de su trabajo de análisis.

Parámetro: IReadOnlyCollection<Token>

Sección de una secuencia de código de la construcción.

Devuelve: `string`

Una cadena vacía o una que contiene “Success” o “Failure”, o una cadena con un objeto JSON conforme al esquema siguiente:

```
{
   “result”:  string
   “message”: string
   “alerts”:
   [
      {
         “description”: string
         “level”:    string
         “name”:    string
         “time”:    unsigned integer
      }
   ],
   “analysisResults”:
   [
      {
         “name”: string
         “time”: unsigned integer
         Any number of additional custom properties
      }
   ],
   “timeSeries”:
   [
      {
         “name”: string
         “unitType”: string
         “unit”: string
         “values”:
         [
            {
```

```
                "time":  unsigned integer
                "value": number
            }
        ],
        "limits":
         [
            {
               "time":        unsigned integer
               "lowerDisplayLimit":  number
               "upperDisplayLimit":  number
               "lowerWarningLimit":  number
               "upperWarningLimit":  number
               "lowerOperatingLimit": number
               "upperOperatingLimit": number
            }
        ]
     }
  ]
}
```

result: string

Indica si el plug-in ha procesado los datos correctamente o no. Debe ser

- el valor literal Success, que indica que el plug-in se ha procesado correctamente, o bien
- el valor literal Failure, que indica que el plug-in ha experimentado un error. El plug-in permanece activo y continúa recibiendo datos.

Esta propiedad es obligatoria.

message: string

Mensaje con más detalles que se adjunta al resultado y se guarda en el registro de auditoría. Esta propiedad es opcional, si no se incluyen datos, no se guarda ningún resultado en el registro de auditoría.

alerts: JSON Array

Matriz que contiene las alertas que el plug-in va a publicar en Renishaw Central durante el proceso. Esta propiedad es opcional, si no se incluye o la matriz está vacía, DataHUB no publica ninguna alerta en Renishaw Central.

alerts[n].description: string

Descripción de los detalles de la alerta. Esta propiedad es obligatoria.

alerts[n].level: string

Uno de los siguientes valores literales indica la importancia de la alerta:

- Info
- Warning
- Error

Esta propiedad es obligatoria.

alerts[n].name: string

Nombre de la alerta. Esta propiedad es obligatoria.

alerts[n].time: unsigned integer

Hora en la que se produce la alerta en microsegundos, relativa a la hora de inicio de la capa. Esta propiedad es obligatoria.

analysisResults: JSON Array

Matriz que contiene los resultados del análisis que plug-in va a publicar en Renishaw Central durante el proceso. Esta propiedad es opcional, si no se incluye o la matriz está vacía, DataHUB no publica ningún resultados del análisis en Renishaw Central.

analysisResults [n].name: string

Nombre del resultado del análisis. Esta propiedad es obligatoria.

analysisResults [n].time: unsigned integer

Hora en la que se produce el resultado del análisis en microsegundos, relativo a la hora de inicio de la capa. Esta propiedad es obligatoria.

analysisResults[n] *Propiedades adicionales*

El resultado del análisis puede incluir cualquier cantidad de propiedades adicionales para almacenar en Renishaw Central, a las que puede acceder con software personalizado. No obstante, no se puede utilizar ninguno de los nombres reservados:

- created
- machine
- source
- type

timeSeries: JSON Array

Matriz que contiene los datos de la serie cronológica que usa el plug-in para publicar los datos definidos en el método *Initialise*. Esta propiedad es opcional, si no se incluye o la matriz está vacía, DataHUB no actualiza ningún dato de la serie cronológica en Renishaw Central.

timeSeries[n].name: string

Nombre de la serie cronológica definido durante la inicialización. Esta propiedad es obligatoria.

timeSeries[n].unitType: string

Tipo de la serie cronológica definido durante la inicialización. Esta propiedad es obligatoria.

timeSeries[n].unit: string

Tipo de la serie cronológica definido durante la inicialización. Esta propiedad es obligatoria.

timeSeries[n].values: JSON Array

Matriz que contiene los pares de series cronológicas que se van a añadir. Una serie cronológica debe incluir la propiedad valores o límites, o ambas.

timeSeries[n].values[m].time: unsigned integer

Hora en la que se produce el valor en microsegundos, relativo a la hora de inicio de la capa. Esta propiedad es obligatoria.

timeSeries[n].values[m].value: number

El valor. Esta propiedad es obligatoria.

timeSeries[n].limits: JSON Array

Matriz que contiene los límites de datos. Hay tres juegos de límites:

1. Operating
2. Warning
3. Display

Las aplicaciones que integran Renishaw Central pueden aplicar estos límites para facilitar la visualización y realizar acciones cuando los puntos de datos de la serie cronológica superan varios de estos límites.

Los límites son efectivos desde la hora asignada por la propiedad hasta la hora asignada por los siguientes límites: los últimos límites definidos son efectivos a perpetuidad.

Si no es necesario que un plug-in facilite los límites de una serie cronológica, es posible que los datos no tengan un alcance de funcionamiento natural o de visualización, por ejemplo. Tampoco es habitual que los límites se modifiquen después de configurarlos, pero pueden modificarse en caso necesario.

Una serie cronológica debe incluir la propiedad valores o límites, o ambas.

timeSeries[n].limits[m].time: unsigned integer

Hora en la que el límite es efectivo en microsegundos, relativo a la hora de inicio de la capa. Esta propiedad es obligatoria.

timeSeries[n].limits[m].lowerDisplayLimit: number

Valor del límite de visualización inferior. Esta propiedad es opcional: si no se especifica ningún valor, el límite de visualización inferior no se modifica.

timeSeries[n].limits[m].upperDisplayLimit: number

Valor del límite de visualización superior. Esta propiedad es opcional: si no se especifica ningún valor, el límite de visualización superior no se modifica.

timeSeries[n].limits[m].lowerWarningLimit: number

Valor del límite de advertencia inferior. Esta propiedad es opcional: si no se especifica ningún valor, el límite de advertencia inferior no se modifica.

timeSeries[n].limits[m].upperWarningLimit: number

Valor del límite de advertencia superior. Esta propiedad es opcional: si no se especifica ningún valor, el límite de advertencia superior no se modifica.

timeSeries[n].limits[m].lowerOperationalLimit: number

Valor del límite de funcionamiento inferior. Esta propiedad es opcional: si no se especifica ningún valor, el límite de funcionamiento inferior no se modifica.

timeSeries[n].limits[m].upperOperationalLimit: number

Valor del límite de funcionamiento superior. Esta propiedad es opcional: si no se especifica ningún valor, el límite de funcionamiento superior no se modifica.

Si la cadena devuelta no cumple el esquema anterior, se trata como un error y se guarda la cadena completa en el registro de auditoría.

6.4.11.9 Método *Shutdown*

El método *Shutdown* se invoca exactamente una vez y la llamada se repite cuando el método *Initialise* devuelve un error, o cuando no hay más datos para procesar (porque se han procesado todos los datos previstos o se ha cancelado el trabajo). Una vez invocado, el plug-in no recibe más llamadas de ningún tipo.

En este método, el plug-in debe realizar el procesamiento final de los datos de construcción, dejar libres los recursos utilizados, etc.

Parámetros: Ninguno

`Devuelve: string`

Si la cadena contiene la palabra “Success”, indica que el plug-in se ha apagado correctamente. En caso contrario, indica que el plug-in ha experimentado un error. En ambos casos, la cadena se guarda completa en los registros de auditoría.

6.4.11.10 Cadena de datos de inicialización

DataHUB inicializa un plug-in V2.0 con una cadena con formato JSON que contiene valores de los parámetros de construcción globales y la cadena de inicialización especificada en el proceso de trabajo de DataHUB Generator para el plug-in de procesamiento del trabajo de la construcción.

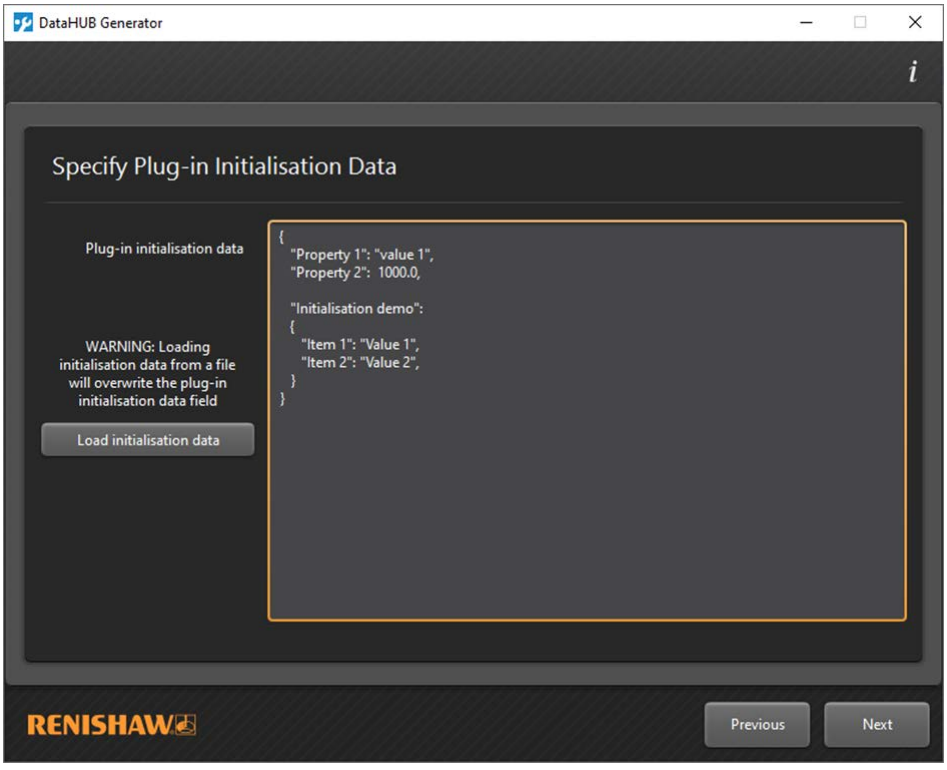


Imagen 42 Cadena JSON del plug-in de entrada

La cadena especificada en el proceso de trabajo de DataHUB Generator también tiene formato JSON. Al añadir propiedades y objetos, puede enviar datos específicos de cada tipo de plug-in data al método *Initialise* de la instancia.

Vea a continuación un ejemplo de la cadena de inicialización del plug-in:

```
{
   “Property 1”: “value 1”,
   “Property 2”:  1000.0,
   “Initialisation demo”:
   {
      “Item 1”: “Value 1”,
      “Item 2”: “Value 2”,
   }
}
```

Consulte la biblioteca de análisis JSON (p.ej., Newtonsoft.Json o System.Text.Json) del plug-in para realizar el proceso de extracción de los valores de la cadena de inicialización, por ejemplo:

```
public string Initialise(string initialisationData)
{
   var instanceData         = JObject.Parse(initialisationData);
   var assemblyFolder       = instanceData[“AssemblyFolder”];
   var outputFolder         = instanceData[“OutputFolder”];
   var numberOfLasers       = instanceData[“NumberOfLasers”];
   var dataHUBGeneratorData = JObject.Parse(instanceData[“InitialisationData”].Value<string>());
   var property1            = dataHUBGeneratorData[“Property 1”];
   var property2            = dataHUBGeneratorData[“Property 2”];
   var item1                = dataHUBGeneratorData[“Initialisation demo”][“Item 1”];
   var item2                = dataHUBGeneratorData[“Initialisation demo”][“Item 2”];
```

Los datos de inicialización de la instancia especificados en DataHUB Generator son también una cadena JSON, por consiguiente, se analiza como JObject antes de acceder a sus propiedades. Los valores de las propiedades `Property1` y `Property2`, y los valores de las propiedades del objeto de demostración anidado `Initialisation` se muestran en la ventana VisualStudio:

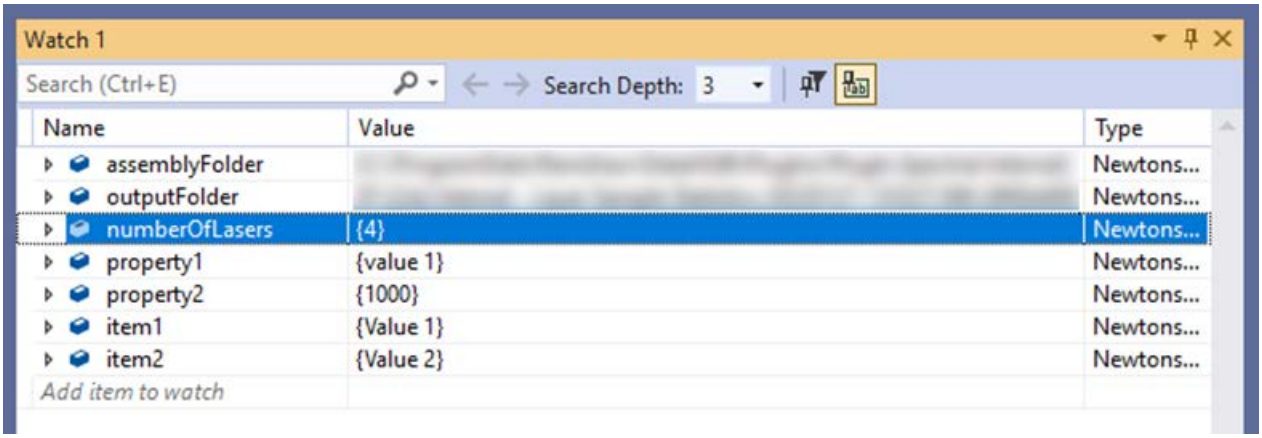


Imagen 43 Cadena de datos de inicialización del plug-in en una llamada a Initialise

Normalmente, hay varios plug-in en curso en un DCPC, por lo que la estrategia recomendada de formato de la cadena de DataHUB Generator consiste en especificar un objeto JSON con nombre por cada tipo de plug-in:

```
{
   “PlugIn.TypeA”:
   {
      “PlugIn parameter 1”:1,
      “PlugIn parameter 2”:2,
   },
   “PlugIn.TypeB”:
  {
      “PlugIn parameter 1”:100,
      “PlugIn parameter 2”:200,
   }
}
```

Para acceder a los datos de inicialización específicos de un plug-in, tiene acceder al objeto anidado correspondiente, por ejemplo, en los plug-in TypeA y TypeB respectivamente:

```
var dataA = dataHUBGeneratorData[“PlugIn.TypeA”];
var dataB = dataHUBGeneratorData[“PlugIn.TypeB”];
```

6.4.11.11 Tipos de código

6.4.11.11.1 DataSourceInformation

El código describe las funciones de un origen de datos de control del proceso.

```
public string      File { get; }
public LayerNumber Layer { get; }
public LaserID     Laser { get; }
public DateTime    LayerStartTime { get; }
public double      MillisecondsPerSample { get; }
public uint        TotalNumberOfSamples { get; }
```

6.4.11.11.2 EndOfLayerData

Este código indica el final de los datos de una capa.

```
public LayerNumber Layer { get; }
public uint        LastSampleTime { get; }
```

6.4.11.11.3 EndOfLayerLaserData

Este código marca el final de las muestras del láser de una capa.

```
public LayerNumber Layer { get; }
public LaserID     Laser { get; }
```

6.4.11.11.4 PacketOfSamples

Este código contiene una colección de muestras.

```
public IreadOnlyList<Sample> Samples { get; }
```

6.4.11.11.5 Sample

El código de la muestra contiene los datos obtenidos en puntos diversos puntualmente durante el proceso de construcción de la capa. Hay gran cantidad de código de muestra por cada capa y cada láser.

- `public uint   Time               { get; }`
- `public double DemandX            { get; }`
- `public double DemandY            { get; }`
- `public double DemandFocus        { get; }`
- `public int    DemandLaserPower   { get; }`
- `public int    LaserModulate      { get; }`
- `public int    MeltVIEWPlasma     { get; }`
- `public int    MeltVIEWMeltpool   { get; }`
- `public int    LaserVIEW          { get; }`
- `public int    LaserOutputPower   { get; }`
- `public int    LaserBackReflection { get; }`

6.4.11.11.6 Separadas

Este código no tiene propiedades.

6.4.11.11.7 StartOfLayerData

Este código marca el inicio de la serie de código correspondiente a una capa.

```
public LayerNumber Layer { get; }
```

6.4.11.11.8 StartOfLayerLaserData

Este código marca el inicio de la serie de código correspondiente al láser de una capa.

```
public LayerNumber Layer { get; }
public LaserID     Laser { get; }
```

6.4.11.11.9 SummaryPacket

Este código contiene valores de resumen producidos por métodos resumidos proporcionados al `filtro SummarisePacketOfSamples`. El juego de valores de resumen se almacena en instancias de la clase anidada `SummaryValues`. Puesto que los métodos de resumen podrían generar resultados fraccionarios, el tipo de cada propiedad de `SummaryValues` es doble.

```
Public uint Time   { get; }
public uint Duration { get; }
public IreadOnlyDictionary<string, SummaryValues> Summary { get; }
public class SummaryValues
{
   public double DemandX             { get; }
   public double DemandY             { get; }
   public double DemandFocus         { get; }
   public double DemandLaserPower    { get; }
   public double MeltVIEWPlasma      { get; }
   public double MeltVIEWMeltpool    { get; }
   public double LaserVIEW           { get; }
   public double LaserOutputPower    { get; }
   public double LaserBackReflectio  { get; }
}
```

6.4.11.11.10 Token

Clase de base de la que se deriva todo el código. Esta clase no contiene datos.

6.4.11.11.11 LaserID y LayerNumber

El tipo subyacente del ID y los números de los láseres es entero, para evitar una alineación incorrecta accidental de, por ejemplo, un número de capa a un ID de láser, estos proporcionan la seguridad de los tipos durante la creación del código anterior.

6.4.11.12 Filtros

La API proporciona un juego de filtros diseñados para realizar tareas comunes al procesar una secuencia de código.

6.4.11.12.1 EmitSampleOnlyIf

Este filtro compara el valor de una entrada de la propiedad de código `Sample`, que emite la muestra `Sample` solo si el valor supera la prueba. Los tipos de código distinto a `Sample` se emiten en la secuencia de salida.

Dispone de los siguientes valores de propiedad:

- DemandX
- DemandY
- DemandFocus
- DemandLaserPower
- MeltVIEWPlasma
- MeltVIEWMeltpool
- LaserVIEW
- LaserOutputPower
- LaserBackReflection
- LaserModulate
- The equality tests are:
- EqualTo
- NotEqualTo
- GreaterThan
- GreaterThanOrEqualTo
- LessThan
- LessThanOrEqualTo
- CustomTest
- IsOn (solo LaserModulate)

La variante final del filtro permite configurar un ensayo de comparación personalizable. Puede utilizar cualquier función que admita dos entradas de enteros y devuelva un Booleano, por ejemplo:

```
var customTest = EmitSampleIf.DemandX.CustomTest(v => v % 2 == 0);
```

Este ensayo personalizado transfiere las muestras con un valor par DemandX.

6.4.11.12.2 SplitWhen

Este filtro compara uno de los valores de propiedad del código de entrada `Sample` con el valor de la propiedad del código precedente y, si es distinto, añade un código `Split` antes de generar el código de entrada, que añade la división `Split` correcta de entrada. Los tipos de código distinto a `Sample` se emiten en la secuencia de salida.

Puede utilizar las alternativas siguientes:

- DemandXChanges
- DemandYChanges
- DemandFocusChanges
- LaserModulateChanges
- DemandLaserPowerChanges

NOTA: Solo se incluyen propiedades de demanda; las demás propiedades de control de procesos son muy variables entre muestras, por tanto, no son adecuadas para dividir la secuencia de código.

6.4.11.12.3 CollateInto

Este filtro agrupa el código `Sample` en paquetes de código `Split` en un único paquete de código `PacketOfSamples`. Se absorbe el código `Split` y `Sample`. El resto de tipos de código se emite en la secuencia de salida.

```
public static Func<Token, IEnumerable<Token>> PacketOfSamples()
```

Esta asignación no tiene ningún valor. Debido a la secuencia de código dividida anteriormente por código `Split` (por ejemplo, por cambios de modulación del láser, posiciones, etc.), la secuencia de salida resultante no contiene `código Sample` o `Split`, ya que se ha reemplazado por código `PacketOfSamples`.

6.4.11.12.4 SummarisePacketOfSamples

Este filtro procesa el código `PacketOfSamples` generado por `CollateInto` y realiza una o varias funciones de resumen en los valores de datos de muestras agrupadas. El constructor de este filtro permite configurar una o varias funciones de resumen denominadas y almacenar los resultados de cada una en el código de salida.

```
public static Func<Token, IEnumerable<Token>> Summarise(params SummaryMethod[] summaryMethods)
```

Construye una instancia SummaryMethod denominada y una función de resumen:

```
public SummaryMethod(string description, Func<IReadOnlyList<int>, double> func)
```

6.4.11.12.5 Pipeline

Para construir filtros en canales, necesita estos dos métodos de ayuda:

```
public static Func<Token, IEnumerable<Token>> First(<Token, IEnumerable<Token>> f1);
public static Func<Token, Ienumerable<Token>> Then(Func<Token, IEnumerable<Token>> f0,
                                                   Func<Token, IEnumerable<Token>> f1);
```

6.4.11.13 Helpers

La API proporciona determinadas clases que definen las operaciones comunes utilizadas en la programación de plug-ins.

6.4.11.13.1 PositionHelpers

Esta clase proporciona funciones de redondeo de valores dobles de manera consistente:

```
public static double RoundToMicrons(double value, double microns);
public static int Round(double value);
```

6.4.11.13.2 Métodos de ayuda de cadenas de retorno

La API proporciona métodos de ayuda para construir las cadenas de retorno necesarias para DataHUB en cada fase de los plug-in (*Initialise*, *Process* y *Shutdown*).

NOTA: DataHUB espera recibir una cadena JSON conforme con la API de cadenas de retorno. Es importante omitir los caracteres especiales con una barra invertida en las cadenas. Si no se incluyen, la cadena JSON puede ser incorrecta y se descarta al enviarla a DataHUB. Por ejemplo:

```
ProcessReturns.Success("this is a badly \"escaped\" message");
ProcessReturns.Success("this is a correctly \\\"escaped\\\" message");
```

Segundo ejemplo:

```
ProcessReturns.Success("this is a badly \nescaped message");
ProcessReturns.Success("this is a correctly \\nescaped message");
```

6.4.11.13.3 Devoluciones de Initialise

Los métodos de ayuda para construir un plug-in individual devuelven cadenas de Initialise a DataHUB como sigue:

```
public static string Success();
public static string Success(string message);
public static string Success(IReadOnlyCollection<string> messages);
public static string Failure();
public static string Failure(string message);
public static string FailureDueToThisException(Exception e);
```

Para crear una o varias series cronológicas en Renishaw Central, puede utilizar el siguiente método de ayuda:

```
public static string SuccessAndCreateTimeSeries(IEnumerable<TimeSeries> timeSeries)
public static string SuccessAndCreateTimeSeries(string message, IEnumerable<TimeSeries> timeSeries)
```

Debe construir un objeto `TimeSeries` con nombre, tipo de unidad y unidad. El generador de objetos `TimeSeries` proporciona una interfaz fluida de tipos de unidades conocidos para facilitar una construcción sencilla y un proceso menos propenso a errores. Vea una demostración en el siguiente ejemplo:

```
var titaniumTimeSeries = TimeSeries.Factory().
                                    Name("Thermal expansion").
                                    Temperature().
                                    Celsius().
                                    Grouping("Titanium").
                                    Create();
```

6.4.11.13.4 Devoluciones de Process

Los métodos de ayuda para construir un plug-in individual devuelven cadenas de *Process* a DataHUB son:

```
public static string Success();
public static string SuccessWithoutInformation();
public static string Success(string message);
public static string Success(IReadOnlyCollection<string> messages);
public static string Failure();
public static string Failure(string message);
public static string FailureDueToThisException(Exception e);
```

Con los siguientes métodos de ayuda, puede usar las cadenas que devuelve el plug-in de la función Process para indicar el éxito o el error, y enviar los datos como actualizaciones de series cronológicas, alertas y resultados de análisis a Renishaw Central.

```
public static string SuccessAndSendData(string message, IEnumerable<UpdateTimeSeries> timeSeries)
public static string SuccessAndSendData(string message, IEnumerable<Alert> alerts)
public static string SuccessAndSendData(string message,
                                        IEnumerable<AnalysisResult>
                                        analysisResults)
public static string SuccessAndSendData(string message,
                                        IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<Alert> alerts)
public static string SuccessAndSendData(string message,
                                        IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<AnalysisResult> analysisResult)
public static string SuccessAndSendData(string message,
                                        IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string SuccessAndSendData(string message,
                                        IEnumerable<UpdateTimeSeries> timeSeries,
```

```
                                        IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string SuccessAndSendData(IEnumerable<UpdateTimeSeries> timeSeries)
public static string SuccessAndSendData(IEnumerable<Alert> alerts)
public static string SuccessAndSendData(IEnumerable<AnalysisResult> analysisResults)
public static string SuccessAndSendData(IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<Alert> alerts)
public static string SuccessAndSendData(IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<AnalysisResult> analysisResult)
public static string SuccessAndSendData(IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string SuccessAndSendData(IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(IEnumerable<UpdateTimeSeries> timeSeries)
public static string FailureAndSendData(IEnumerable<Alert> alerts)
public static string FailureAndSendData(IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<Alert> alerts)
public static string FailureAndSendData(IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(string message, IEnumerable<UpdateTimeSeries> timeSeries)
public static string FailureAndSendData(string message, IEnumerable<Alert> alerts)
public static string FailureAndSendData(string message, IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(string message,
                                        IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<Alert> alerts)
public static string FailureAndSendData(string message,
                                        IEnumerable<UpdateTimeSeries> timeSeries,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(string message,
                                        IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
public static string FailureAndSendData(string message,
                                        IEnumerable<UpdateTimeSeries> timeSeries,
```

```
                                        IEnumerable<Alert> alerts,
                                        IEnumerable<AnalysisResult> analysisResults)
```

El objeto `TimeSeries` definido durante *Initialise* puede actualizarse con valores, límites o ambos. El objeto `TimeSeries` proporciona un método Update que devuelve un objeto `UpdateTimeSeries`, como muestra el siguiente ejemplo:

```
var update = titaniumTimeSeries.
      Update().
      WithValues((0, 32), (100, 67)).
      WithLimits(TimeSeriesLimit.Factory().
                                  Time(0).
                                  LowerDisplayLimit(0).
                                  NoUpperDisplayLimit().
                                  LowerWarningLimit(30).
                                  NoUpperWarningLimit().
                                  LowerOperatingLimit(10).
                                  NoUpperOperatingLimit());
```

Pare generar alertas en Renishaw Central, puede construir un objeto `Alert` y transferirlo al método de ayuda `SendData` apropiado. El gestor `Alert` proporciona una interfaz de construcción sencilla, como muestra el ejemplo siguiente:

```
var overheatingAlert = Alert.Factory().
                            Descripción ("La temperatura de la pieza ha superado el umbral").
                            Warning().
                            Name("Overheating").
                            Time(350);
```

Por último, para activar resultados de análisis en Renishaw Central, puede construir un objeto `AnalysisResult` y transferirlo al método de ayuda SendData apropiado. El gestor `AnalysisResult` proporciona una interfaz de construcción sencilla, como muestra el ejemplo siguiente:

```
var analysisResult = AnalysisResult.Factory().
                                    Name("Thermal expansion rate").
                                    Time(250).
                                    AddAdditionalProperty("Rate", 0.05).
                                    Create();
```

6.4.11.13.5 Devoluciones de Shutdown

Los métodos de ayuda para construir un plug-in sencillo devuelven cadenas de *Shutdown* a DataHUB como sigue:

```
public static string Success();
public static string Success(string message);
public static string Success(IReadOnlyCollection<string> messages);
public static string Failure();
public static string Failure(string message);
public static string FailureDueToThisException(Exception e);
```

6.5 Plug-in de exportación de paquetes

El hardware LaserVIEW y MeltVIEW obtiene muestras de datos a 100 kHz, con muestras diferenciadas que contienen distintos valores. Aunque esta resolución es útil para varias tareas de análisis, una representación de resolución más baja de las mismas muestras podría ser más adecuada para muchas tareas de análisis, por lo que el plug-in agrupa los datos en “paquetes” definidos por:

- el láser disparando y
- una demanda de las posiciones X e Y constante.

Esta definición agrupa efectivamente el juego de muestras durante una exposición sencilla del láser en la mesa del polvo. Por cada juego, se producen los valores promedio y mediano de los valores del sensor de control de procesos de Laser/MeltVIEW. Cabe señalar que, debido a las características de funcionamiento del láser, las muestras al inicio de una exposición pueden contener valores de Laser/MeltVIEW ligeramente más bajos. Por consiguiente, el valor mediano podría ser un resumen más representativo de los datos de la muestra que el valor promedio.

6.5.1 Ejecución del plug-in

Para usar este plug-in, ejecute DataHUB Generator y seleccione la carpeta *Build Data* que contiene los archivos de control de procesos. Seleccione una carpeta para los archivos de salida del plug-in. Si es necesario (por ejemplo, el archivo de información de la construcción *BuildStarted aaaa.mm.dd. HH.mm.ss.dhxml* no se encuentra en la carpeta de entrada), compruebe que el *Número de láseres* es correcto para la construcción, ya que el plug-in no procesa los datos si no coincide el número de láseres seleccionado con el número real instalado en el hardware LaserVIEW/MeltVIEW de la máquina de FA que ha generado los datos. Los dos valores restantes, *Separación de capas* y *Altura de construcción* pueden dejarse sin modificar, ya que no son necesarios para el plug-in.

Escriba una descripción y pulse Siguiente.

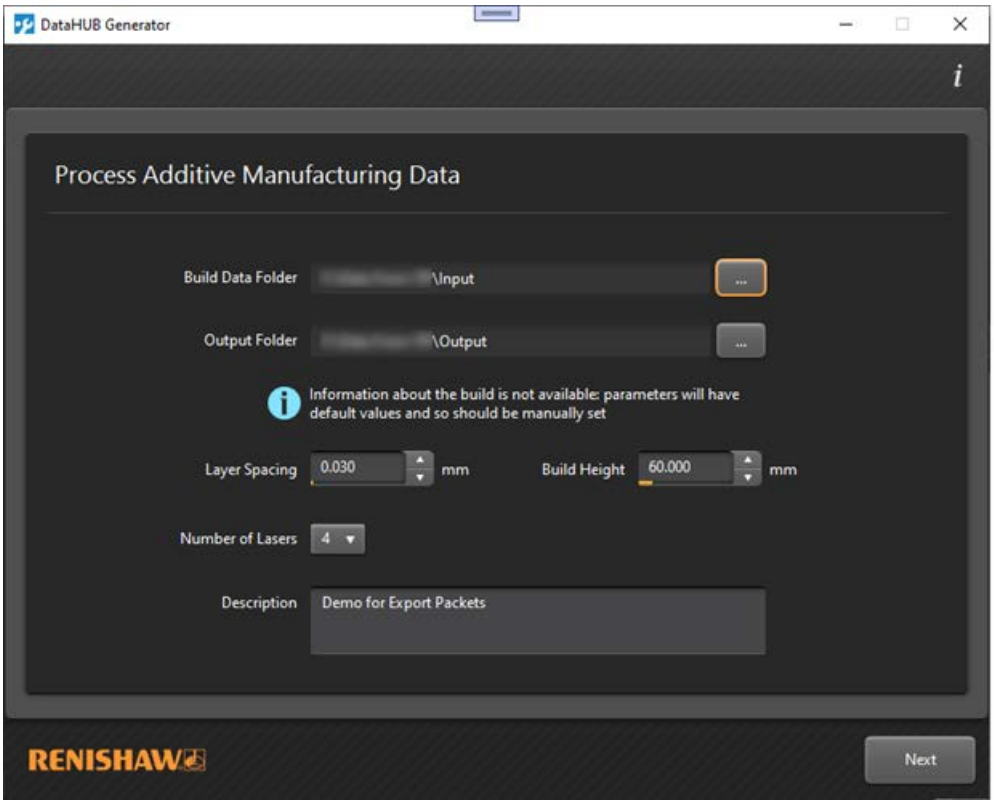


Imagen 44 Fase del proceso de trabajo 1: configuración de datos

En la siguiente parte del proceso de trabajo, seleccione “Procesar datos LaserVIEW y MeltVIEW data mediante plug-in” y pulse Siguiente.

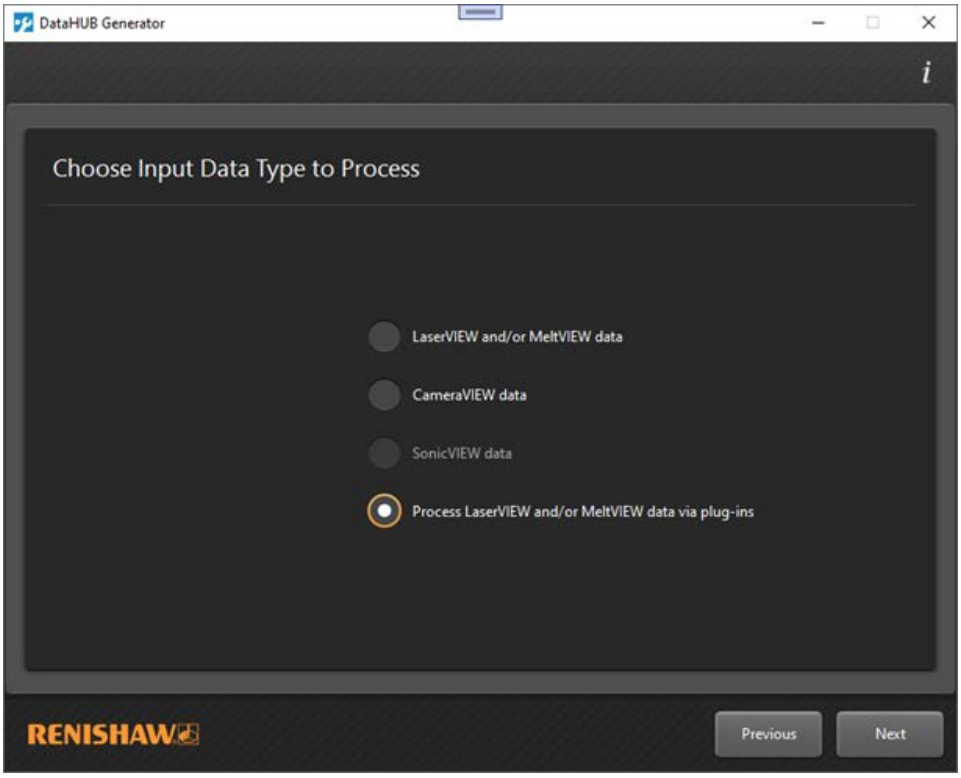


Imagen 45 Fase del proceso de trabajo 2: selección de procesamiento del plug-in

Puesto que este plug-in no requiere inicialización de datos, puede dejar el cuadro de texto vacío. Pulse Siguiente.

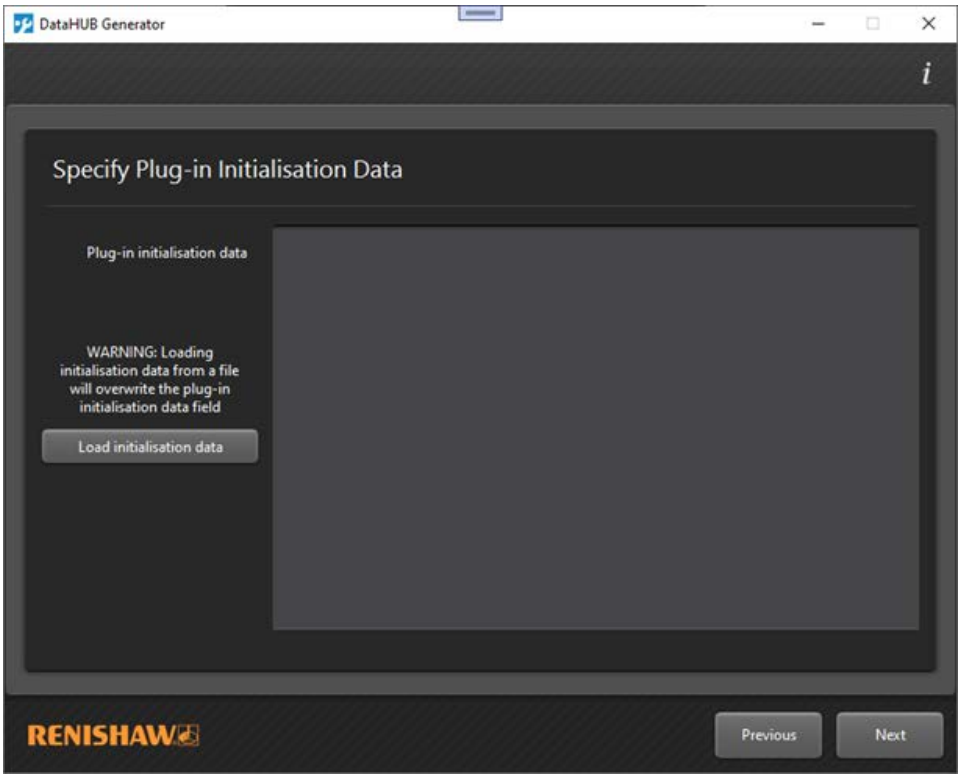


Imagen 46 Fase del proceso de trabajo 3: inicialización del plug-in

En la parte final del proceso de trabajo, pulse “Iniciar el procesamiento del plug-in LaserVIEW/MeltVIEW”.

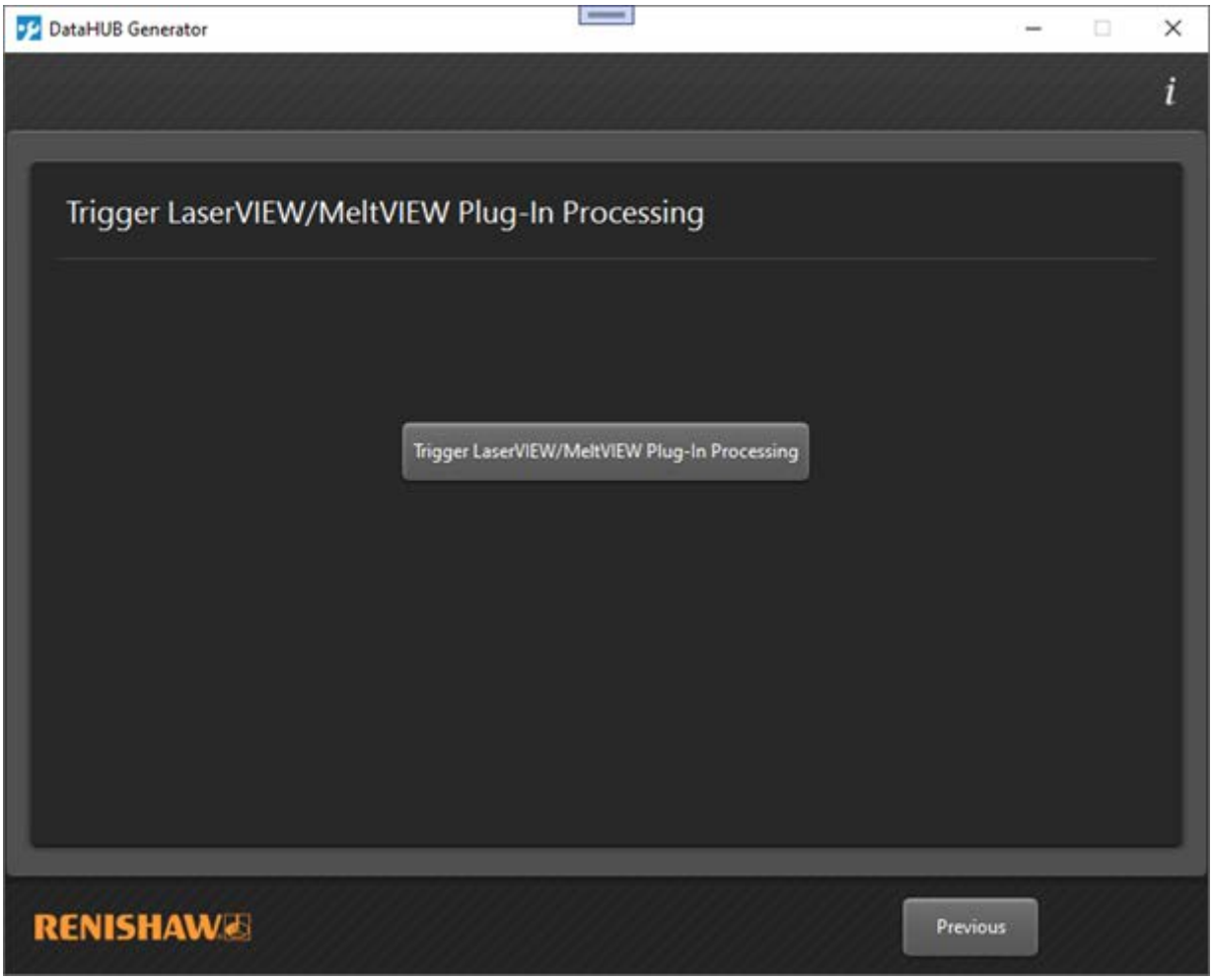


Imagen 47 Fase del proceso de trabajo 4: iniciar el procesamiento

Puede salir de la aplicación DataHUB Generator.

6.5.2 Nombre de archivo y ubicación

La carpeta de salida *Output* especificada en el proceso de trabajo es la carpeta en la que el plug-in almacena los archivos. El plug-in crea una carpeta secundaria denominada como se indica a continuación:

[2] Export Packets aaaaMMdd-HHmmss.ff (GUID)

Donde “aaaaMMdd-HHmmss.ff” es la fecha y hora de inicio del procesamiento de datos del plug-in y “GUID” es un código exclusivo. Se ha elegido este nombre para asegurar que, si se repite la ejecución del plug-in con los mismos datos, no se sobrescriben otros guardados en la carpeta.

En la carpeta, el plug-in guarda un archivo por cada láser y cada capa, denominado “Packet data for layer x, laser y.txt”, donde x es el número de capa e y el número de láser. Aunque el archivo utiliza la extensión “txt”, tiene formato delimitado por tabulador, por lo que puede importarse en muchos paquetes de software sin conversión.

6.5.3 Contenido del archivo

Cada archivo empieza con una fila de títulos de columna:

- Hora de inicio: relativa al inicio de la capa (µs)
- Duración: conforme a los tiempos de las primeras y últimas muestras agrupadas en el paquete (µs)
- Demanda X: posición del paquete en la mesa del polvo (izquierda –125 a derecha +125 mm)
- Demanda Y: posición del paquete en la mesa del polvo (delante –125 a atrás +125 mm)
- Demanda de enfoque: enfoque del rayo láser en la mesa del polvo (±5 mm)
- Demanda de polvo láser (promedio): valores promedio de las muestras agrupadas en el paquete
- Plasma MeltVIEW (promedio): valores promedio de las muestras incluidas en el paquete
- Baño de fusión MeltVIEW (promedio): valores promedio de las muestras incluidas en el paquete
- LaserVIEW (promedio): valores promedio de las muestras incluidas en el paquete
- Retrorreflexión del láser (promedio): valores promedio de las muestras incluidas en el paquete
- Potencia de salida del láser (promedio): valores promedio de las muestras incluidas en el paquete
- Demanda de potencia del láser (medio): valor medio de las muestras incluidas en el paquete
- Plasma MeltVIEW (medio): valor medio de las muestras incluidas en el paquete
- Baño de fusión MeltVIEW (medio): valor medio de las muestras incluidas en el paquete
- LaserVIEW (medio): valor medio de las muestras incluidas en el paquete
- Retrorreflexión del láser (medio): valor medio de las muestras incluidas en el paquete
- Potencia de salida del láser (medio): valor medio de las muestras incluidas en el paquete

A continuación, incluye una serie de filas que contienen los valores resumidos de cada paquete de datos de AMPM:

Start time	Duration	Demand X	Demand Y	Demand focus	Demand laser power (mean)	MeltVIEW plasma (mean)	MeltVIEW melt pool (mean)	LaserVIEW (mean)	Laser back reflection (mean)	Laser output power (mean)	Demand laser power (median)	MeltVIEW plasma (median)	MeltVIEW melt pool (median)	LaserVIEW (median)	Laser back reflection (median)	Laser output power (median)
29800	20	-110.365	-76.35	0	2418	59.5	59	167	12801	16216	2418	59.5	59	167	12801	16216
29830	20	-110.36	-76.325	0	2418	631.5	371	696.5	17184	19679.5	2418	631.5	371	696.5	17184	19679.5
29860	20	-110.365	-76.29	0	2418	1017	672	725.5	17709.5	20158.5	2418	1017	672	725.5	17709.5	20158.5
29890	20	-110.365	-76.265	0	2418	902	653.5	729.5	17847.5	20287.5	2418	902	653.5	729.5	17847.5	20287.5
29920	20	-110.37	-76.235	0	2418	962.5	746.5	735	9562	7149	2418	962.5	746.5	735	9562	7149
30970	20	-110.26	-76.14	0	2418	379.5	434	424	15568.5	18501	2418	379.5	434	424	15568.5	18501
31000	20	-110.265	-76.165	0	2418	717.5	432.5	725.5	17634.5	20103	2418	717.5	432.5	725.5	17634.5	20103
31030	20	-110.265	-76.19	0	2418	1022	684	730	17855	20304.5	2418	1022	684	730	17855	20304.5
31060	20	-110.27	-76.22	0	2418	964	725.5	737.5	17891.5	20339.5	2418	964	725.5	737.5	17891.5	20339.5
31090	20	-110.27	-76.245	0	2418	975	756.5	735	17934.5	20400.5	2418	975	756.5	735	17934.5	20400.5
31120	20	-110.27	-76.275	0	2418	1180.5	913.5	739	17960.5	20414	2418	1180.5	913.5	739	17960.5	20414
31150	20	-110.265	-76.3	0	2418	1196	922.5	736	17984	20421	2418	1196	922.5	736	17984	20421
31180	20	-110.265	-76.33	0	2418	1163	888	740.5	17965.5	20420	2418	1163	888	740.5	17965.5	20420
31210	20	-110.265	-76.36	0	2418	1077.5	853	734.5	17973	20416	2418	1077.5	853	734.5	17973	20416
31240	20	-110.26	-76.385	0	2418	1098.5	875.5	743.5	17963.5	20403.5	2418	1098.5	875.5	743.5	17963.5	20403.5
31270	20	-110.265	-76.415	0	2418	1165.5	947	737	17928	20363	2418	1165.5	947	737	17928	20363
31300	20	-110.265	-76.445	0	2418	1086	873.5	741.5	9582	7164	2418	1086	873.5	741.5	9582	7164
32350	20	-110.175	-76.535	0	2418	418	456	428	15556.5	18509	2418	418	456	428	15556.5	18509
32380	20	-110.175	-76.505	0	2418	834	490	728	17640	20111	2418	834	490	728	17640	20111
32410	20	-110.175	-76.475	0	2418	1060	711	742	17858	20319	2418	1060	711	742	17858	20319
32440	20	-110.175	-76.45	0	2418	1125	812	741.5	17918	20383	2418	1125	812	741.5	17918	20383
32470	20	-110.165	-76.42	0	2418	1210	894.5	744.5	17959.5	20409.5	2418	1210	894.5	744.5	17959.5	20409.5
32500	20	-110.17	-76.39	0	2418	1202	860	745	17952	20387	2418	1202	860	745	17952	20387

Esta página se ha dejado intencionadamente en blanco.

RENISHAW
apply innovation™

www.renishaw.es/contacto

#renishaw

+34 93 663 34 20 **spain@renishaw.com**

Nº de referencia: H-5800-6858-01-A
Edición: 10.2023